Abstraction techniques for scalable soft error analysis and mitigation

Adrian Evans

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Micro et Nano Électronique**

Arrêté ministériel : 7 août 2006

Présentée par

Adrian EVANS

Thèse dirigée par **Michael NICOLAIDIS**

préparée au sein du **laboratoire TIMA**
dans **l'École Doctorale Électronique, Électrotechnique, Automatique et Traitement du signal**

Techniques d'abstraction pour l'analyse et la mitigation des effets dus à la radiation

Thèse soutenue publiquement le **19 juin 2014**,
devant le jury composé de :

Professeur Dominique HOUZET
INPG (Laboratoire GIPSA, Grenoble), Président
Dr. Jean ARLAT
CNRS (LAAS, Toulouse), Rapporteur
Professeur Luis ENTRENA ARRONTES
Universidad Carlos III de Madrid, Rapporteur
Dr. Michael NICOLAIDIS
CNRS (Laboratoire TIMA, Grenoble), Directeur de thèse

Acknowledgments

Undertaking a PhD has been an exciting and challenging experience and I have learned a great deal about the topic matter as well as about scientific research in general. I am indebted to numerous people; more than I can fully acknowledge here.

First and foremost, I wish to thank my thesis supervisor, Michael Nicoloaidis, who welcomed me warmly when I arrived at the TIMA laboratory. Over the last three years, his guidance and insight have been invaluable. He has also taught me a great deal about the importance of protecting intellectual property.

I wish to express my thanks to Jean Arlat for his contributions to the RIIF workshop and to him, Luis Entrena and Dominique Houzet for serving on my thesis committee and providing their feedback.

To Dan Alexandrescu, CEO of iROC Technologies, I am beholden for his encouragement, for granting me time to write my thesis and for helping me to obtain numerous reference materials. To the entire staff at iROC Technologies, I wish to express my thanks for providing a warm environment where I have learned immensely from experts in the field of soft errors. I especially thank Enrico Costenaro with whom I collaborated on the work on the analysis of transient errors and Jocelyne Baudoin who helped manage the time I spent working on my thesis.

The first year of my thesis was spent at the TIMA Laboratory and I wish to express my thanks to Dominique Borrione, director of the laboratory. Mounir Benabdenbi helped me with my course work and Lorena Anghel was of great assistance with the organization of the RIIF Workshop. Laurence Ben Tito and Anne-Laure Fourneret-Itie helped me navigate the administrative procedures.

Furthermore, I wish to express my thanks to Shi-Jie Wen of Cisco Systems who encouraged many parts of the work in this thesis. His help in gaining access to actual designs for the portion of the work on selective flip-flop mitigation and his support for extending the work on combinatorial SET analysis were extremely valuable.

The work related to RIIF was highly collaborative. I would like to thank Oliver Bringmann who was the co-chair of the RIIF workshop as well as Pedro Reviriego Vasallo and Alfonso Sánchez Macián of the University of Nebrija who helped develop the parser and test cases for RIIF.

The work on Bloom Filters was a collaboration with Salvatore Potarelli and Marco Ottavi at the University of Rome "Tor Vergata". The technical interaction was very enjoyable and I wish to acknowledge the Median Cost Action which covered my travel expenses for several trips.

In addition, I would like to thank Li Chen and Hao Xie of the University of Saskatchewan who lead the implementation of the work on redundant logic.

Last, but certainly not least, I express my sincere thanks to my wife Anne and my children Zoë and Ben who have been extremely patient while I have spent many hours in front of the computer. Anne's precious help in proofreading my manuscript is sincerely appreciated.

Contents

Glossary

ACE Architecturally Correct Execution. 56, 57

ACL Access Control List. 130, 131, 152

ADD Algebraic Decision Diagram. 47, 57–59, 92

ALU Arithmetic Logic Unit. 56, 90, 96, 100–103

ASIC Application Specific Integrated Circuit. 48, 52, 63, 65, 124, 125

AVF Architectural Vulnerability Factor. 46, 56, 62, 110, 114, 122

BDD Binary Decision Diagram. 47, 57, 58, 92, 140

BFM Bus Functional Model. 50

BICS Built-In Current Sensor. 2, 150–152

BPSG Boron Doped Phosphosilicate Glass. 6

CAM Content Addressable Memory. 127, 128, 130, 132, 133

CED Concurrent Error Detection. 140

CEU Code Emulated Upset. 53–55

CLA Carry Lookahead Adder. 91

CRAM Configuration RAM. 149–152

CRC Cyclical Redundancy Check. 68, 123, 124

DAG Directed Acyclical Graph. 120

DICE Dual-Interlocked storage Cell. 20, 22, 23, 40, 42

DPM Defects per Million. 48

DSTB Douple Sampling with Time Borrowing. 31, 32, 42, 43

DUE Detected Uncorrected Error. 47, 63, 107, 114, 116

DUT Device Under Test. 49, 50

DVS Dynamic Voltage Scaling. 21, 27

ECC Error Correcting Code. 16, 18, 54, 63, 65, 94, 95, 112, 114, 118, 127, 150, 152

EDA Electronic Design Automation. 90, 122, 126

EDC Error Detection Correction. 35, 36, 41, 42

EDM Error Detection Mechanism. 54, 55

EDR Electrical De-Rating. 13, 46, 58, 91–93, 97, 100, 102, 103, 125

EM Electro-Migration. 126

EPP Error Propagation Probability. 91, 101, 142, 143

FDD First-level Dynamically Dead. 56

FDR Functional De-Rating. 13, 14, 46, 48, 50, 61, 62

FDSOI Fully Depleted Silicon on Insulator. 15, 123

FI Fault Injection. 54

FIFO First-In First Out. 65, 66

FIT Failure in Time. 11, 14, 15, 90, 93, 96, 99–102, 105–108, 114, 115, 117, 141, 145–147

FPGA Field Programmable Gate Array. 38, 46, 51–53, 149–152

FR Fault Reduction. 142–144

HCI Hot Carrier Injection. 20, 126

HDL Hardware Description Language. 79

HPC High Performance Computing. 78

HSEET Hierarchical Soft Error Estimation Tool. 92

IC Integrated Circuit. 107, 108, 122

IoT Internet of Things. 16

IP Intellectual Property. 107, 114, 115

LDR Logical De-Rating. 46, 51, 56–58, 60, 91–93, 95, 97, 100–102, 107

LUT Look Up Table. 52

MBE Multiple Bit Error. 112

MBU Multiple Bit Upset. 40, 63, 106, 110, 112, 114, 117

MDR Memory De-Rating. 46, 115

MoRV Modelling Reliability under Variability. 126

MOS Metal Oxide Semiconductor. 8

MTBF Mean Time Between Failure. 108

NBTI Negative Bias Temperature Inversion. 20, 126

NDA Non-Disclosure Agreement. 47

NMI Non-Maskable Interrupt. 53

NP Network Processor. 47, 63, 73, 77

OO Object Oriented. 117, 122

PIPB Propagation Induced Pulse Broadening. 13

PLA Programmable Logic Array. 140

PLL Phase Locked Loop. 8

PVF Program Vulnerability Factor. 46, 56

PVT Process, Voltage, Temperature. 30

QoS Quality of Service. 48, 127, 130, 131

RAID Redundant Array of Independent Disks. 110

RIIF Reliability Information Interchange Format. 2, 3, 47, 90, 103, 106, 108–112, 114, 116–118, 120–122, 125, 126

RTL Register Transfer Language. 3, 46, 51, 53, 57, 63, 66, 67, 78, 79, 89, 90, 92–94, 96, 100, 101, 103

SBU Single Bit Upset. 106, 109, 110, 112, 114, 117, 118

SDC Silent Data Corruption. 47, 63, 78, 109, 110, 114, 116, 118, 125

SDF Standard Delay Format. 52, 58, 62, 100, 102

SEB Single Event Burnout. 8

SEC Single Error Correct. 118

SECDED Single Error Correct, Double Error Detect. 109, 110, 112, 114

SEE Single Event Effect. 150

SEFI Single Event Functional Interrupt. 8, 106

SEGR Single Event Gate Rupture. 8

SEL Single Event Latchup. 8

SER Soft Error Rate. 7–10, 15, 16, 18, 21–23, 28, 63, 73, 76, 89–95, 100, 102, 103, 106, 107, 125, 128

SET Single Event Transient. 2, 3, 7, 8, 10–14, 18, 20–27, 29–32, 34–36, 38, 40, 42, 43, 47, 49, 52, 53, 57, 58, 61, 62, 89–93, 95, 96, 100–102, 125, 141, 147

SEU Single Event Upset. 7, 8, 10–14, 20–22, 24–26, 28–30, 32, 34–40, 42, 43, 49, 52, 53, 56, 57, 59, 61, 63–66, 68–70, 78, 80, 83, 87, 102, 110, 114, 123, 125, 127, 134, 150

SEUT Single Event Upset Tolerant. 23

SFI Statistical Fault Injection. 71, 77, 79

SHE Single Bit Hard Error. 118

SoC System on Chip. 47, 107, 108, 114, 122, 123

SOI Silicon on Insulator. 4

SOP Sum of Products. 140, 146, 147

STA Static Timing Analysis. 63

SWIFI Software Implemented Fault Injection. 53–56

TCAD Technology CAD. 93, 106

TCAM Ternary Content Addressable Memory. 2, 106, 127, 128, 130–138, 152

TD Transition Detector. 36, 37

TDR Temporal De-Rating. 11, 12, 23, 24, 42, 46, 57, 58, 61–63, 70, 91–93, 97, 100, 102

TDTB Transition Detector with Time Borrowing. 31, 32, 42

TFIT Transistor FIT. A tool developed by IROC Technologies to estimate the SER sensitivity of individual cells. These can be memory cells, sequential cells or combinatorial cells. 93

TID Total Ionizing Dose. 9

TMR Triple Modular Redundancy. 38, 83

TVF Temporal Vulnerability Factor. 11

UCLI Unified Command Line Interface. 67

ULA Ultra Low Alpha. 3

VPI Verilog Procedural Interface. 67, 78

Chapter 1

Introduction

Contents

1.1 Introduction to Soft Errors and System Reliability

Electronic systems based on complex silicon devices play an increasingly critical role in our lives and it is extremely important that they function reliably. Faults induced by the effects of natural radiation are a significant source of failure and there exists a vast body of research devoted to analyzing and mitigating these effects. Much of the academic work has focussed on achieving accurate results when analyzing relatively small circuits. In industry, the size of circuits continues to grow and integrated circuits with over ten million flip-flops are very common.

The focus of this thesis is on soft error analysis and mitigation techniques for very large circuits. The main contributions consist of improvements in fault-injection simulations, a technique to identify critical flip-flops in large circuits, a technique to perform a high-level analysis of the effect of Single Event Transients (SETs) and a new modeling language, called Reliability Information Interchange Format (RIIF), which can be used to model the propagation of faults through complex circuits.

The manuscript is organized as follows. The first chapter provides an overview of soft error effects, including a review of their causes, a brief history of their discovery and a discussion of the basic masking mechanisms. Chapter 2 presents an overview and comparison of the numerous hardened latch and flip-flop designs which have been proposed to mitigate faults. This review includes cells targeted to mitigate both radiation-induced faults and timing faults. Although timing faults are fundamentally different from radiation induced faults, the choice to include both was intentional as there are many similarities between the techniques to mitigate transients and those to mitigate timing faults. It is well known that the vast majority of faults do not propagate and in chapter 3, a review of the existing techniques to analyze fault propagation is presented. The second half of chapter 3 presents a case study of the analysis of soft error effects for three large design blocks from a network processor.

In chapter 4, a technique to identify functionally critical flip-flops in large designs is presented. The technique is based on grouping together similar flip-flops in order to reduce the complexity of the analysis. The results of applying these techniques to the design blocks studied in chapter 3 are then presented. Chapter 5 describes a hierarchical approach for quickly computing the effect of SETs in large combinatorial circuits. Then, in chapter 6, a new modeling language called RIIF is described. This language is designed to provide a standard means to specify the rate of occurrence of faults as well as how faults propagate in complex circuits. This is followed, in chapter 7, by the conclusions and a discussion of plans for future work.

In addition to the main topics of the thesis, additional work performed in the context of three collaborative projects is presented in the appendices. In appendix A and appendix C, two different approaches to protect a special type of memory, Ternary Content Addressable Memorys (TCAMs), against errors are described. The first technique is based on using a parallel data-structure called a Bloom Filter. The second technique is based on using a Built-In Current Sensor (BICS) to detect the actual radiation induced fault when it occurs. In appendix B, an algorithm to

synthesize approximate logic functions that can mask both transient and permanent faults is presented.

Overall, the work presented in this thesis seeks to apply abstraction in order to enable coarser grained and hierarchical analysis of the effect of faults. The initial work in chapter 4 shows how large sets of flip-flops can be grouped together in order to simplify the process of selective mitigation. In chapter 5, the focus shifts to SETs and a hierarchical analysis approach is presented which can be applied at the Register Transfer Language (RTL) or even architectural level. Finally, the RIIF modeling language provides a framework for combining failure models for smaller blocks into complete models for chips or even systems. Taken together, these techniques facilitate the analysis and mitigation of radiation induced faults in complex integrated circuits.

1.2 Soft Error Effects

1.2.1 Sources of Natural Radiation

In terrestrial applications, the three main sources of energetic particles that induce soft error effects are alpha particles, high-energy neutrons and thermal neutrons [Baumann 2005]. New work has recently highlighted an emerging risk of upsets induced directly by muons [Sierawski 2011, Ibe 2012] and potentially in the future from electrons and gamma rays. In space applications, the radiative environment is much harsher and soft errors can be induced directly by high energy cosmic particles including protons and heavy nuclei.

1.2.1.1 Alpha Particles

An alpha particle has the structure of a helium atom nucleus, consisting of two protons and two neutrons. Alpha particles typically have energies in the range of 4-9 MeV and travel at low speeds (5% of the speed of light). Alpha particles are created through the decay of unstable isotopes such as ^{238}U, ^{232}Th, ^{210}Po, ^{210}Pb and ^{212}Pb. Trace amounts of these isotopes may be found in the packaging materials that are in close contact to the die, typically in the solder bumps, ceramic package materials or aluminum interconnect. These particles do not penetrate very deeply into materials; in silicon, the penetration of a 10MeV alpha particle is less than 100μm, thus only the materials that are in direct contact with the die are of concern [Baumann 2005].

It has become common practice to select Ultra Low Alpha (ULA) packaging materials for integrated circuits used in applications where reliability is a concern. ULA implies an alpha emission rate below $0.002\alpha/cm^2/hour$. Undesirable elements, such as uranium, can be chemically removed. However, removing specific isotopes (e.g. ^{210}Pb) of an element such as lead is challenging and this is sometimes achieved by 'aging' the materials in order to allow sufficient time for the unstable isotopes to decay. The fact that the unstable isotopes decay over time, makes the measurement

of alpha emissivity very challenging. For example, ^{210}Pb has lower alpha emissivity than ^{210}Po, one of the by-products on its decay chain. This evolution of alpha emissivity over time makes it difficult to implement consistent quality controls.

1.2.1.2 Cosmic Radiation

The second source of soft errors results from the interactions between cosmic rays and the atmosphere. In outer space, there are primary cosmic rays consisting of [Heald 2005] :

- 92% protons
- 6% alpha particles
- 1% heavy nuclei

In the outer atmosphere, these particles have a flux of approximately $1600\,m^2/s$ and energies up to 100 GeV [Ziegler 1996]. These particles interact with the earth's atmosphere producing complex chains of secondary and tertiary particles. At ground level, the flux of particles consists of muons, protons, neutrons and pions [Ziegler 1981]. The typical spectrum of neutron flux at ground level as well as at some accelerated test facilities is reproduced from [JEDEC 2006] in figure 1.1. It is important to note that this spectrum varies significantly based on altitude and location due to the effect of the earth's magnetic field. A comprehensive study of the neutron spectrum at ground level was performed by [Gordon 2004, Gordon 2005] and for higher altitudes by [Normand 1996].

The earth's atmosphere blocks most cosmic particles, however, when these particles strike atoms in the atmosphere, they produce high-energy neutrons which can travel freely through air. When these high-energy neutrons strike the nucleus of a silicon atom, they can produce either elastic or inelastic collisions. Most elastic collisions result in only a small amount of energy being released. Inelastic collisions result in a fission reaction and in two or more recoil fragments which can include protons, neutrons, ions and heavy recoil nuclei. This can result in the deposition of a large amount of charge in a small volume [Seifert 2010a]. Figure 1.2 illustrate the collision of a neutron with a silicon nuclei and lists some of the more frequently occurring by-products. A full study of neutron-silicon interactions is presented in [Wrobel 2000]. Although the majority of soft error effects are induced by interactions between neutrons and silicon, the interactions with other atoms including oxygen in Silicon on Insulator (SOI) technologies must be taken into account [Wrobel 2003].

1.2.1.3 Thermal Neutrons

The third source of terrestrial soft errors is from low energy neutrons ($E_n \ll 1MeV$). These are neutrons that have lost kinetic energy until they reach a state where they are in thermal equilibrium with their environment. Certain nuclear fission reactions

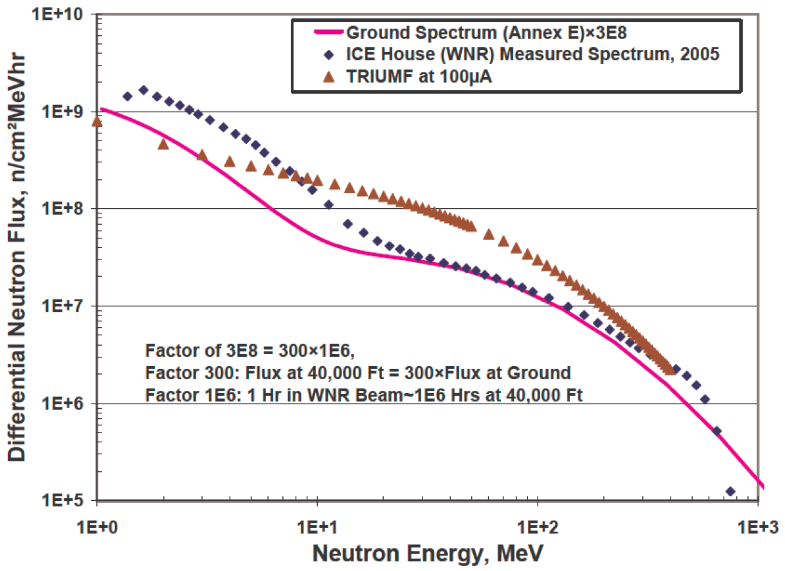


Figure 1.1: Terrestrial Neutron Spectrum - Actual and Test Facilities

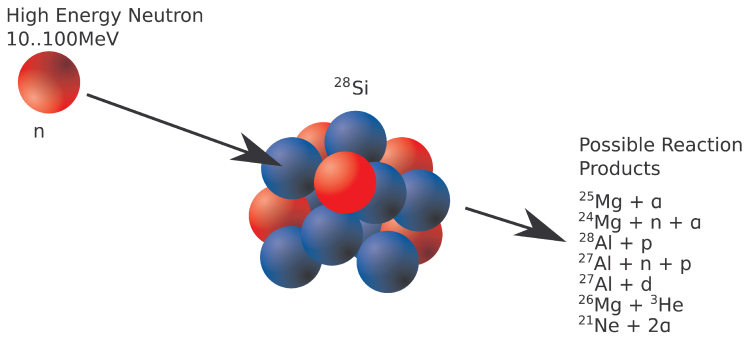


Figure 1.2: Recoil Ions from Neutron/Silicon Interaction

become much more probable with these low-energy neutrons and result in reactions yielding charged particles. The most common such reaction is with the $^{10}\mathbf{B}$ isotope of boron. Boron is used as a p-type dopant and is also used as an implant in insulating layers formed of Boron Doped Phosphosilicate Glass (BPSG). The natural distribution of the two boron isotopes is $^{11}\mathbf{B}$ (80.1%) and $^{10}\mathbf{B}$ (19.9%), however, BPSG typically has a higher fraction of $^{10}\mathbf{B}$. When a low energy neutron strikes $^{10}\mathbf{B}$, it results in an alpha particle (1.47 MeV) and a lithium nucleus (equation 1.1). Both of these cause charge to be deposited and can induce soft errors [Baumann 2005].

$$^{10}\mathbf{B} + \mathbf{n} \longrightarrow {}^{14}\mathbf{Li} + \alpha \tag{1.1}$$

Recent work on thermal neutrons [Wen 2010a, Wen 2010b, Zhang 2011, Fang 2013] has shown that even when BPSG is not used in the fabrication process, devices can have a sensitivity to thermal neutrons. These works also highlight the challenges in consistently testing the sensitivity to thermal neutrons due to the lack of a standard test procedure analogous to [JEDEC 2006].

1.2.2 Brief History of Soft Error Effects

A proper understanding of the effects of radiation on electronics started to emerge in the late 1970's and the history is well outlined in [Ziegler 1996] and [Nicolaidis 2011]. As early as the 1950's, there had been anecdotal reports of problems with electronic equipment during nuclear weapons tests. The electronics used in early satellites were known to be unreliable and redundancy thus error detection had become common practice. The first publication about the effect of space radiation on satellites was published in [Binder 1975].

In the following year, the first study was published on ground-level soft error effects observed in a Cray-1 computer and this paper was recently republished in [Normand 2010]. In the following two years, errors due to alpha-particles in DRAMs were studied by Intel [May 1978, May 1979]. It is interesting to note that these first observations were on DRAM cells. Today, however, due to technology scaling, DRAM cells are among the most robust types of storage cells.

At about the same time, Ziegler (IBM) and Lanford (Yale) realized that nuclear interactions between cosmic rays and silicon materials might cause secondary particles and trigger errors. These results were first published in [Ziegler 1979]. Later in 1979, Kolasinski (Aerospace Corporation) performed the first heavy-ion tests on electronics at Lawrence Berkeley Laboratory using iron and kyrpton ions [Kolasinski 1979] to simulate the radiative conditions experienced by satellites. In the same year, [Guenzer 1979] published the first results from accelerated neutron testing and provided an improved analysis of the interactions between neutrons and silicon.

Already in 1979, researchers at IBM had started to develop a framework for modeling the diffusion and collection of charge induced by energetic particles. By 1983, IBM had started to publish on SEMM, their internal Monte Carlo tool for

simulating radiation effects [Sai-Halasz 1983]. The second generation of this tool (SEMM2) is still in use by IBM today [Tang 2004].

About this time, there were high-profile incidents at Intel and at IBM [Ziegler 1996] where radioactive contamination of production materials resulted in memory devices with excessively high soft error rates. The publicity associated with these events increased the awareness of soft error phenomena both in industry and in the academic community.

In 1983, the first evidence of soft errors produced by muons and pions was published [Dicello 1983]. Around the same time, internal studies by IBM showed that Soft Error Rate (SER) was dependent on altitude. These included studies using special test-boards, as well as the analysis of the logs of systems installed at high-altitude locations such as Denver, Colorado [Ziegler 1996]. In the following years, IBM continued to test 25 different chips under neutron and proton beams and studied the effect of voltage, temperature, angle of incidence, process variation and logic-state. The comprehensive results were compiled in an internal IBM report published in 1986.

In the early 1990's IBM developed the first ion micro beam which made it possible to study the effect of charge deposition at exact positions on a chip with an accuracy of approximately 1 micron [Geppert 1991, Heidel 1993]. A few years later, Baumann demonstrated the first evidence of soft errors induced by thermal neutrons [Baumann 1995].

From this brief outline of the early research into soft error effects, it can be seen that all of the fundamental concepts related to the physics of soft error phenomena as well as the accelerated test methodologies that are used today have been well understood for nearly twenty years. Currently, the outstanding challenges relate to computing the net effect of soft errors in very large circuits which is the focus of the research in this thesis.

1.2.3 Taxonomy of Radiation Effects

When an ionizing particle strikes a circuit, it can produce many different effects depending on the type of circuit, the energy of the particle and the nature of the strike. This section provides an overview of the different effects.

1.2.3.1 SEUs

A Single Event Upset (SEU) is an inversion in the stored value in a flip-flop, latch or memory cell as the result of radiation induced charge. In this work, the term SEU refers only to an upset that *directly* occurs in a sequential cell. Note that in some work [Shuler 2009], the authors also use the term SEU to refer to a SET that has been sampled in a flip-flop or latch.

1.2.3.2 SETs

A Single Event Transient (SET) occurs when a radioactive event causes charge to be deposited around a combinatorial logic gate producing a current pulse. For this pulse to become an error, it must propagate through the combinatorial network and then arrive at the input of a sequential element just at the sampling point. Due to the low probability of a pulse being captured, the majority of SETs do not propagate and until recently it has been assumed that for terrestrial applications, the overall SER contribution from SETs is very minor.

1.2.3.3 SEFI

Single Event Functional Interrupt (SEFI) is a broad term that refers to the loss of some major functionality in a complex device due to a radiation induced event. Typically, a SEFI is caused by a SEU in critical control logic. The effect of a SEFI is detectable and it does not result in permanent damage. A SEFI can be recovered by resetting the device.

1.2.3.4 SEL

A Single Event Latchup (SEL) event occurs when the charge deposited by an energetic particle causes the parasitic thyristor (PNPN), formed by the P-drain, N-well, P-well and N-drain, to be triggered, creating a short circuit path from power to ground (see figure 1.3 and 1.4). Once latchup has occurred, the only way to restore normal operation is to cut the power to the circuit. Typically, a SEL event results in increased current consumption and if this current is too high, the effect can be destructive. A destructive SEL is sometimes referred to as a Single Event Burnout (SEB). Decreasing operating voltage makes modern process technologies less sensitive and even immune to SEL.

1.2.3.5 SEGR

Single Event Gate Rupture (SEGR) occurs in MOS power transistors. When the additional carriers produced by the strike of an energetic particle are accelerated through the electric field, the field is increased, potentially causing the dielectric to breakdown and the transistor to be permanently damaged. SEGR is of growing concern as power transistors are being increasingly used in electric vehicles.

1.2.3.6 Clock Network Faults

Energetic particles can affect the operation of circuitry in the clock distribution network of a chip. This includes PLLs [Sondon 2013, Fujita 2014] as well as in the buffers in the clock tree [Seifert 2005]. In the latter work, the authors identify two main effects due to upsets in the clock network : radiation-induced jitter and radiation-induced race. Jitter occurs if an upset causes the clock edge to move forward causing a setup violation. A race condition occurs if an upset causes a latch

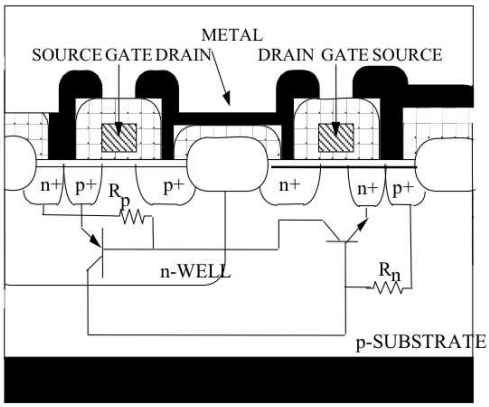


Figure 1.3: Parasitic Thyristor (PNPN)

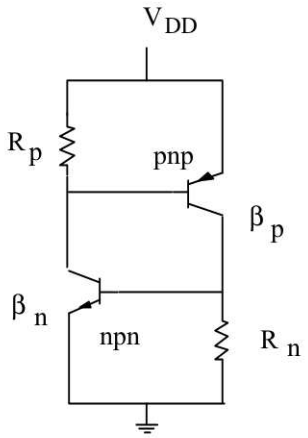


Figure 1.4: Thyristor Formed by PNPN Junction

that is closed to become open allowing data to 'race' through to the next stage. The authors find that the contribution of radiation induced race can be on the order of 20% of the total effective SER for a full chip. Simulation based analysis of the clock network in smaller designs [Ebrahimi 2014] shows the contribution from the clock network to be much smaller. In both works, however, it is reported that the SER contribution of the clock network is dominated by the buffers at the leaf level of the clock tree.

1.2.3.7 Power Supplies

Beyond standard digital logic, radiation effects can perturb the operation of peripheral circuits including switching power supplies. In [Ren 2014], it is shown that the band-gap reference circuit in a particular commercial power "brick" is very sensitive to radiation. Under alpha, neutron and laser radiation upsets to the band-gap reference would cause the power monitor to trigger a shut-down of the power supply. When performing a system-level SER analysis, it is important to consider the potential risk due to the power supplies, especially since these faults typically produce high-impact outages.

1.2.3.8 Dose Effects

Electronics used in space or highly radioactive environments may be degraded due to the cumulative effect of exposure to radiation. The first mechanism is due to the Total Ionizing Dose (TID) which results from charge being trapped in the oxide layer and causing a shift in the switching characteristics of the transistor. A second mechanism is caused by the displacement of silicon atoms in the regular crystalline

structure. This also affects the electrical characteristics. Eventually, the parametric shift can result in a circuit failure.

1.2.4 Summary of Radiation Effects

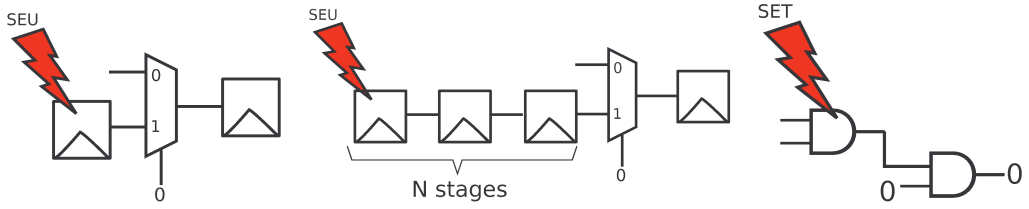
In large digital circuits, the majority of the failures are normally caused by SEUs and SETs and the focus of this thesis is on analyzing these effects. However, it is important to note, that when analyzing the overall SER sensitivity of a system, all of the radiation-induced failure mechanisms must be considered.

1.3 Masking Effects

Fortunately, not all radiation induced faults propagate and produce errors because of the numerous masking effects. The raw rate of faults can be *de-rated* to obtain an effective error rate using a *de-rating* factor. In this work, a *de-rating* factor of '1' indicates that *all* faults propagate and a value of '0' indicates that all faults are *blocked*. The definitions are important as some authors use the term *masking factor* to indicate the fraction of faults that are *masked*, which is the opposite of a *de-rating factor*. In the following sections, the basic masking mechanisms are defined. The first half of chapter 3 contains a review of analysis techniques for evaluating the effects of fault masking on circuits.

1.3.1 Logical Masking

A fault is logically masked if it can not propagate through the gates in a combinatorial network based on their logic function. An SEU can be logically masked in the cycle when it occurs (figure 1.5(a)) or it may be logically masked several cycles later (figure 1.5(b)). Of course, SETs can also be logically masked as shown in figure 1.5(c). It is important to note that an analysis of logical masking requires only a representation of the circuit and does not require any knowledge about the function.

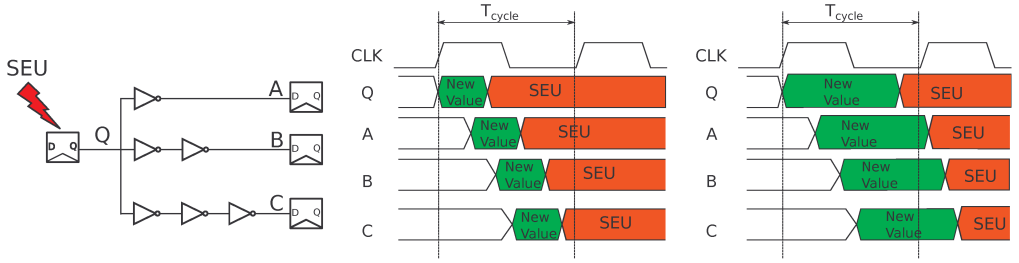


(a) Logical Masking : SEU (b) Logical Masking : SEU after N clocks (c) Logical Masking : SET

Figure 1.5: Logical Masking of SEUs and SETs

1.3.2 Temporal De-Rating

In synchronous digital circuits, in order for faults to propagate, they must be sampled by sequential elements. For an SEU in a flip-flop to be sampled, it must occur sufficiently early in the clock period in order for the erroneous value to meet the setup time of one or more downstream flip-flops. Both the cases of a masked and unmasked fault are shown in figure 1.6



(a) SEU in a Flip-Flop (b) SEU Sampled Downstream (c) SEU is Temporally Masked

Figure 1.6: Temporal Masking of SEUs

Intuitively, it is clear that as the available slack time on the paths increases, so does the probability of an SEU being captured. A first approximation is that the temporal de-rating factor for SEUs varies with the ratio of the slack to the clock period. Of course, if the arrival of the SEU causes a setup or hold violation, then it is uncertain whether the fault is captured. In [Nicolaidis 2011, chapter 5], the probability of capture is assumed to be proportional to the extent of overlap of the SEU with the setup-hold window and with this assumption, the Temporal De-Rating (TDR) is given by equation 1.2.

$$TDR_{SEU} = \frac{t_{slack} + \frac{t_{setup}}{2} - \frac{t_{hold}}{2}}{T_{clock}} \tag{1.2}$$

An in-depth study of the temporal masking of SEUs is presented in [Seifert 2004]. The authors show the results of fault injections performed with SPICE at different points in time through the clock period [1]. A graph of the observed results for latches and flip-flops is reproduced in figure 1.7. Even when there is no delay in the logic (e.g. $t_{prop} \approx 0$), there is still some temporal de-rating due to intrinsic delay of the sequential element [2].

SETs are also subject to temporal masking although this is also referred to as *latch window masking* and is shown in figure 1.8. To a first approximation, the TDR

[1] In work by Intel, the term Temporal Vulnerability Factor (TVF) is used instead of TDR, although the meaning is the same

[2] The master and the slave latches in a flip-flop are each only vulnerable during half of the clock period. Sometimes this intrinsic de-rating of 50% is embedded in the reported FIT rate. In other cases, as in the cited work, this 50% factor is included in the TDR or TVF factor. Thus, in figure 1.7, an abscissa value of 0.5 represents no temporal masking.

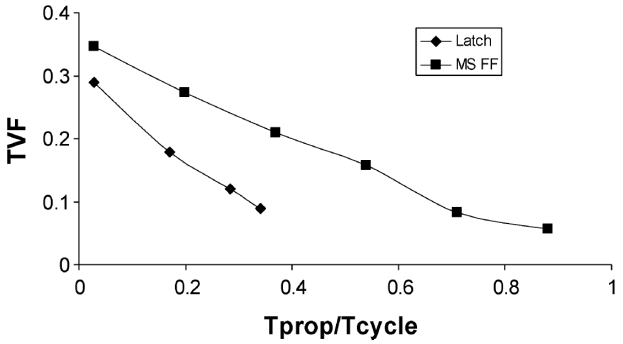
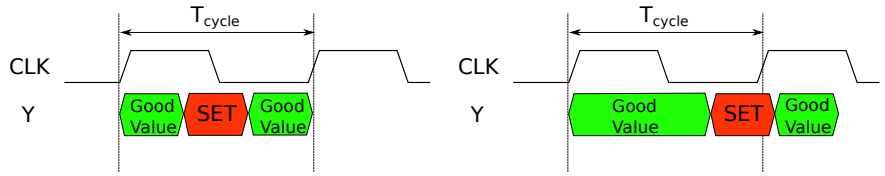


Figure 1.7: Simulated Results of TDR in Latches and Flip-Flops [Seifert 2004]

is proportional to the ratio of the induced pulse width to the clock period as shown in equation 1.3. Extensive test results have shown that at higher frequencies, the TDR factor increases and the effect of SETs is more severe [Nguyen 2005, Gill 2009, Mahatme 2011].



(a) SET is Masked (b) SET is not Masked

Figure 1.8: Temporal Masking of SETs

$$TDR_{SET} = \frac{\int_{w=minPW}^{w=maxPW} w \, \mathrm{d}w}{T_{clk}} \tag{1.3}$$

In figure 1.9, all of the possible alignments of a pulse compared to the sampling clock edge are shown, including both the case when the pulse is longer and when it is shorter than the setup-hold window. For the cases of $PW > t_{setup} + t_{hold}$, the *overlaping width*, OW, is defined to be the extent that the pulse lies within the setup-hold window. As with the SEU analysis, the error capture probability is assumed to be proportional to the ratio of OW to the full setup-hold window: $\frac{OW}{t_{hold}+t_{setup}}$. For the cases of $PW < t_{setup}+t_{hold}$, the two violation cases are considered together, and the error latching probability is also taken to be linear with the overlapping width ratio as before. Since the arrival time of SETs is uniform over time, both cases can be averaged over a full clock period. The calulated error probablity for each case is shown in table 1.1 and it turns out that the overall error latching probability across a full clock cycle is given by PW/T_{clk}.

It is important to note that in this analysis, the pulse width of interest is that at the input to the sampling flip-flop. The shape of a radiation induced pulse may be

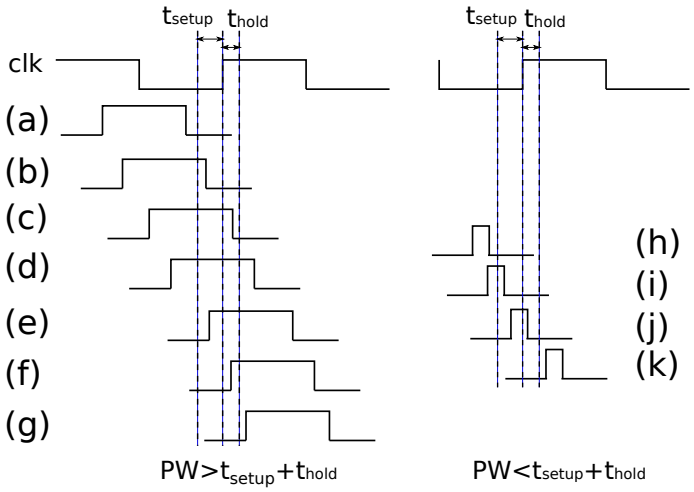


Figure 1.9: SET Pulse Alignment Cases

distorted as it propagates through a combinatorial network. This effect is referred to as Propagation Induced Pulse Broadening (PIPB) and has been extensively studied in [Cavrois 2008].

1.3.3 Electrical Masking

Digital logic is based on signals crossing a switching threshold to discriminate between zero and one values. When a weak SET is created, the voltage on the affected node may not cause the switching threshold to be crossed and the fault is thus masked electrically. Due to their inherent capacitance and limited slew rate, all digital logic gates inherently act as low pass filters, blocking very narrow pulses. This ability to block narrow SETs is referred to as Electrical De-Rating (EDR). The effect of EDR is best analyzed using SPICE simulations where the true, post-layout capacitances have been included, however, in practice this is not feasible for large circuits. EDR can also be modelled in digital simulators by associating an inertial delay to individual gates.

1.3.4 Functional Masking

It is quite possible for a SEU or SET to significantly change the state sequence of a circuit, however, due to the function of the actual application, the effect may be benign. Some obvious examples, would be an SEU that causes a single pixel in a video stream to be modified or that causes a very small delay in the delivery of packet data, as shown in figure 1.10. If we consider the example of a delayed packet, on the output pins of the device, at the vector level, the output sequences may appear highly divergent, despite the fact that the functional effect is very minor.

The additional de-rating provided by the function of the circuit is referred to as Functional De-Rating (FDR). As will be seen in the case study presented in

Table 1.1: SET Pulse Alignment and Capture Probabilities

Pulse Width	Case	Case Error Probability	Comments
$PW > t_{setup} + t_{hold}$	a	0.0	correct value latched
	b	$\frac{1}{T_{clk}} \cdot \int_0^{t_{setup}} \frac{OW}{t_{hold}+t_{setup}} \mathrm{d}OW$	set-up time violation
	c	$\frac{1}{T_{clk}} \cdot \int_{t_{setup}}^{t_{setup}+t_h} \frac{OW}{t_{hold}+t_{setup}} \mathrm{d}OW$	hold time violation
	d	$\frac{PW-t_{setup}-t_{hold}}{T_{clk}}$	wrong value latched
	e	$\frac{1}{T_{clk}} \cdot \int_{t_{hold}}^{t_{setup}+t_h} \frac{OW}{t_{hold}+t_{setup}} \mathrm{d}OW$	set-up time violation
	f	$\frac{1}{T_{clk}} \cdot \int_0^{t_h} \frac{OW}{t_{hold}+t_{setup}} \mathrm{d}OW$	hold time violation
	g	0.0	correct value latched
$PW < t_{setup} + t_{hold}$	h	0.0	correct value latched
	i	$2 \cdot \frac{1}{T_{clk}} \cdot \int_0^{PW} \frac{OW}{t_{hold}+t_{setup}} \mathrm{d}OW$	metastability, partially overlapped
	j	$\frac{PW \cdot (t_{hold}+t_{setup}-PW)}{T_{clk} \cdot (t_{hold}+t_{setup})}$	metastability, totally overlapped
	k	0.0	correct value latched
Overall	-	$\frac{PW}{T_{clk}}$	-

section 3.2, this de-rating is significant and must not be ignored. Unlike the other de-ratings, FDR does require an understanding of the application and requires that the effects of errors be divided into classes based on their system-level severity.

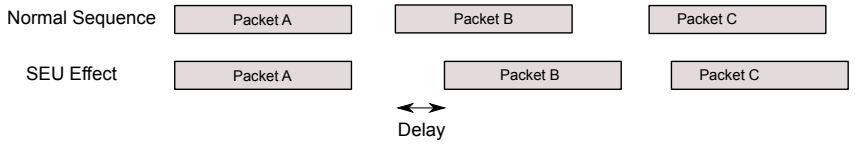


Figure 1.10: SEU Causing a Minor Delay in Packet Delivery

1.3.5 Computing the Overall SER

Given that the intrinsic technology FIT rates and de-rating factors are available, then a full chip FIT rate can be calculated based on the following equations:

$$SER_{chip} = SER_{sequential} + SER_{combo} + SER_{clock} \tag{1.4}$$

$$SER_{sequential} = \sum_{i \in FF} FIT_i^{SEU} \cdot LDR(i) \cdot TDR(i) \tag{1.5}$$

$$SER_{combo} = \sum_{i \in G} \int_{w=min}^{w=max} FIT_i^{SET}(w) \cdot TDR_i(w) \cdot LDR_i \cdot EDR_i(w) \, \mathrm{d}w \tag{1.6}$$

In the above equations, FIT_i^{SEU} is the intrinsic rate of occurrence of SEUs for the given flip-flop instance and $FIT_i^{SET}(w)$ is the rate of occurrence of SETs of width w in the instance of the specific combinatorial gate. The above equations

only hold for a given set of input vectors or workload. In practice, the SER must be evaluated for a representative subset of the actual workloads.

In the above equations, the de-rating factors are assumed to be independent of each other and are thus simply multiplied. In practice, this assumption is made in industrial SER analysis [Nguyen 2005]. In reality a given fault may experience different masking effects along different paths, as illustrated by the example in figure 1.11. Some authors [Miskov-Zivanov 2010] make the claim that treating de-rating factors independently can introduce significant errors, however, their claim is based on considering specific paths on small benchmark circuits. In fact, the amount of error that is introduced by considering de-rating factors independently, when averaged across large circuits, remains an open question. Recent work [Ebrahimi 2014] has shown how different de-rating factors can be accurately evaluated in a hybrid simulation and emulation based fault-injection platform.

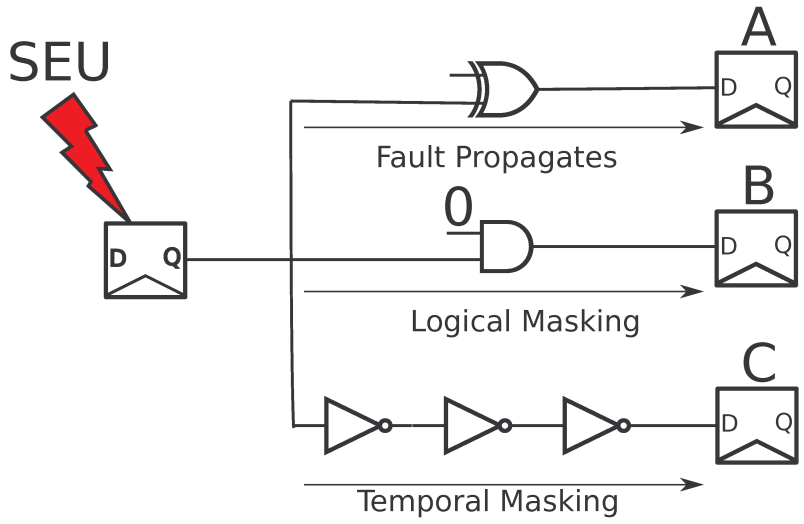


Figure 1.11: Different Masking Factors on Different Paths

1.4 Current SER Trends

Understanding whether the overall impact of soft errors is increasing, decreasing or generally stable is important and is a topic that is addressed in numerous publications [Shivakumar 2002, Baumann 2002, Ibe 2010, Slayman 2011, Dixit 2011]. These works present the measured or simulated soft error rates in memory cells or flip-flops over several technology nodes. Care must be taken when extrapolating past trends, as changes in process technology may be disruptive. Recent data [Seifert 2012] shows that 22nm FINFET devices are about 3x less sensitive than equivalent planar devices and the reported FIT rates for 28nm FDSOI is below 10 FIT/Mbit [Roche 2013]. New threats from muons and electrons have been

raised [Ibe 2012, Sierawski 2011], however, it does not appear that these particles significantly contribute to the SER at current technology nodes. At some point, the critical charge may become sufficiently small that this will change.

Independent of the per-cell SER, certain broad trends can be identified:

1. The total number of integrated circuits is growing driven by larger data centers, the Internet of Things (IoT) and the proliferation of mobile devices.
2. The total number of transistors per integrated circuits continues to grow (Moore's Law [Moore 1965]).
3. Computers are increasingly used in *critical* applications such as autonomous driving and in medical devices.
4. Operating frequencies are roughly constant and performance increases are coming from additional parallelism (more gates).
5. The terrestrial radiative environment ($\approx 14\ n/cm^2/hr$) remains constant although the sensitivity of devices to different particles (e.g. alpha, muons, protons) may evolve.

To illustrate the impact of the growing number of transistors per die, in figure 1.12, we plot the transistor counts for recent processors based on data gathered by [Wikipedia 2014]. Based on the assumption that 90% of the die area of a processor is covered by cache-memory and that there are approximately 16 transistors in a latch, we can estimate the latch counts. From this graph, it appears that designs with a hundred million latches will appear by 2016. Even if the per-cell SER rate continues to decrease, given the broad trends (points 1..3 identified above), it appears that radiation-induced soft errors will continue to be a concern.

1.4.1 Combinatorial SER

The SER threat from memories has been well known for over fourty years and can be effectively mitigated using Error Correcting Code (ECC). As will be seen in chapter 2, there exist many cell designs for SER hardened flip-flops and using techniques such as those presented in chapter 4, it is possible to selectively harden flip-flop and thus manage the sequential component of SER.

Historically, the contribution of soft errors from combinatorial logic has been quite small, due to the numerous de-rating factors that were outlined in section 1.3. However, given that memory and flip-flop SER can be well managed, the relative contribution of combinatorial SER is increasing, as illustrated in figure 1.13. In a well known paper [Shivakumar 2002], it was predicted that by the 50nm technology node, combinatorial SER would equal sequential SER. It appears that this prediction was over-stated, however, more recent research [Mahatme 2011, Ebrahimi 2014] continues to highlight this trend.

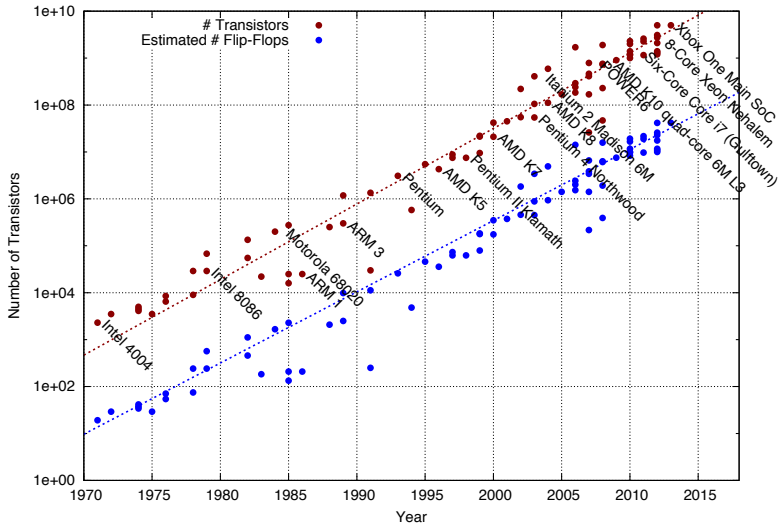


Figure 1.12: Over 40 Years's of Moore's Law

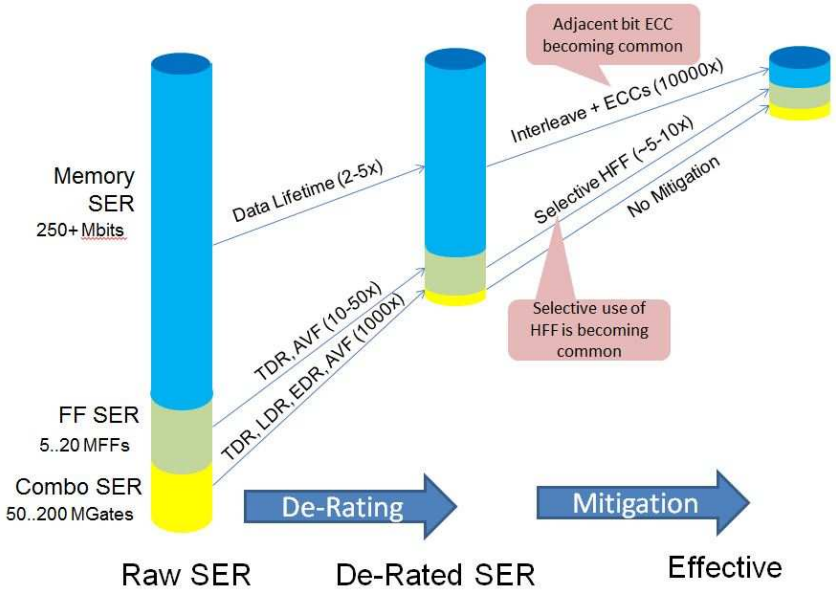


Figure 1.13: Trends in Combinatorial Logic

During accelerated radiation testing of complex devices such as processors, it is difficult to differentiate the sequential and combinatorial components. Due to the complexity of the de-rating analysis for combinatorial SER, it is also difficult to accurately compute the combinatorial contribution at the full-chip level. The methodology presented in chapter 5 describes a new approach to estimate the contribution of SETs.

This chapter has provided a brief overview of the causes of soft errors and the basic mechanisms that prevent their propagation. Assuming ECC is used to protect memories, then the largest contribution to the overall SER comes from flip-flops or latches. In the following chapter, we present a survey of the SER hardened sequentials that have been proposed in the literature.

Chapter 2

Robust Sequentials

Contents

2.1 Introduction

In the early 1990s, due to the growing awareness of soft errors, there was interest in developing flip-flop and latch designs that were resistant to radiation-induced upsets. Some of the earliest designs were the HIT cell [Bessot 1993] and the Dual-Interlocked storage Cell (DICE) [Calin 1996]. The DICE circuit remains the basis for many hardened flip-flops. Layout optimizations have been proposed and are typically applied in industrial DICE implementations to provide increased robustness (e.g. LEAP-DICE [Lee 2010]).

In space applications, SETs have historically been a concern and there is evidence that they are also becoming an issue in high-end terrestrial applications [Mahatme 2011, Evans 2013]. Mitigation of SETs is generally more complex than protection against SEUs and there exist many techniques including those presented in appendix B. Although transients occur in combinatorial logic, one way to to protect against them is to *detect* when they reach the input of a flip-flop or latch. Special sequential cells can be designed to detect, and even filter, transients when seen their at their data input.

Starting around the year 2000, there was interest in sequential cells that could detect timing violations. Multiple factors drove this new interest, including emerging concerns about the parametric degradation of logic cells through aging effects such as Negative Bias Temperature Inversion (NBTI) and Hot Carrier Injection (HCI) which result in increased transistor switching delays. In industry, the standard approach to mitigate aging effects had been to introduce additional margins into the timing budget. Smaller process geometries fundamentally result in increased variation as the switching characteristics of transistors are now affected by defects consisting of only a few atoms. Furthermore, the increase in the number of transistors per die, requires tighter statistical variance in order to achieve a given level of yield or reliability per die. These factors are driving up the timing margins that must be budgeted to guarantee devices operate correctly over their entire lifetime. It is, however, possible to design sequential cells that can detect timing violations.

By detecting timing violations during operation, the voltage or frequency can be dynamically adjusted to enable correct circuit operation without the need for large, up-front timing margins.

Finally, as concerns about power dissipation have continued to grow, the ability to use Dynamic Voltage Scaling (DVS) to reduce voltage, and benefit from the quadratic reduction in active power, has become attractive. The use of sequential cells that can detect timing violations allows DVS to dynamically adapt to the circuit's operating conditions including the workload.

To summarize, sequential cells can be designed to mitigate three classes of faults:

(i) *SEUs* - Upsets that directly affect the storage node or the clock nodes within the storage cell.

(ii) *SETs* - Upsets in the upstream combinatorial logic that propagate to the input of a sequential cell.

(iii) *Timing Faults* - Increased delay in the upstream combinatorial logic causing the setup constraints to be missed and incorrect data to be sampled.

To further complicate matters, in the event of any of the above classes of errors, the goal may be to simply detect and signal its occurrence or it may be to actually correct or mask the error.

In section 2.2, we review some of the better known protected sequential designs. Each design is presented and its ability to mitigate each of the three types of faults is analyzed. This analysis includes a careful review of the additional overheads and system constraints that are imposed by the design.

The focus of this chapter is on circuit-level techniques for hardened sequentials. In many cases, transistor layout plays a critical role in the level of SER protection of the cell and some data about the extent of the impact of layout is presented. However, it is beyond the scope of this thesis to provide an in-depth study of layout techniques.

Certain sequentials are able to detect errors but not correct them. How a given system recovers from a detected error quickly becomes very application specific. Certain techniques, such as instruction replay and pipe-line counterflow, are commonly used in processor pipelines. In this chapter, the emphasis is on the circuit-level error detection mechanisms and not on the architectural or micro-architectural recovery.

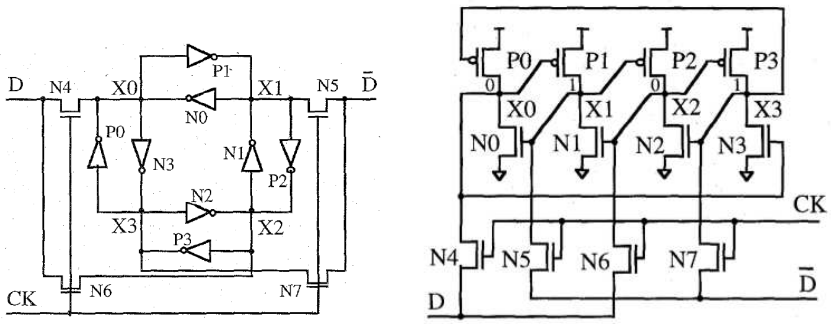
It is important to note that all of the robust sequentials that are described in this chapter rely on two basic principles. Robustness is achieved either through spatial redundancy, temporal redundancy or a combination of the two. These basic principles have been well understood for a long time and their application to robust sequentials was described in detail in [Nicolaidis 1999]. It remains important, however, to study each proposed design in detail, as there is significant variation in the implementation cost and robustness, as will be seen.

There are fundamental trade-offs between area, performance and radiation robustness [Rennie 2012]. In section 2.3, we compare the different designs that were studied based on the types of faults they are able to mitigate and the penalties that they introduce (area, timing and to a lesser extent power).

2.2 Survey of Hardened Sequentials

2.2.1 DICE

The DICE cell is well known and the circuit diagram is reproduced from [Calin 1996] in figure 2.1. The design consists of redundant latches which are coupled together to form a feedback loop. If an upset occurs at any of the storage nodes ($X_0..X_3$), it will propagate only in one direction, depending on its polarity, but the state of the upset node is restored by the feedback from the other direction. In this way, the DICE cell provides strong immunity against an upset in any single storage node. Of course, the DICE was not designed to provide detection of SETs or timing faults.



(a) Circuit Diagram (b) Transistor Level

Figure 2.1: Circuit Diagrams of DICE Cell

2.2.1.1 SEU Robustness

The DICE cell does provide SEU robustness, however, there are intrinsic limitations to its SEU resilience. The first order concern is the fact that if two sensitive nodes are struck, the stored value can be upset [Baze 2008]. Even in 90nm technologies, multi-node upsets were a concern. This problem can be mitigated by increasing the separation between the sensitive nodes and test results presented in [Seifert 2010b] show that SER decreases exponentially with node separation. Figure 2.2 is reproduced from [Seifert 2010b] and illustrates this effect[1].

[Hazucha 2003], and more recently in [Berg 2013], point out one of the main shortcomings of the DICE design. When an upset occurs, it takes some time for the

[1]Due to the test methodology, the reported data only includes upsets to the storage nodes (i.e. clock node upsets are not considered).

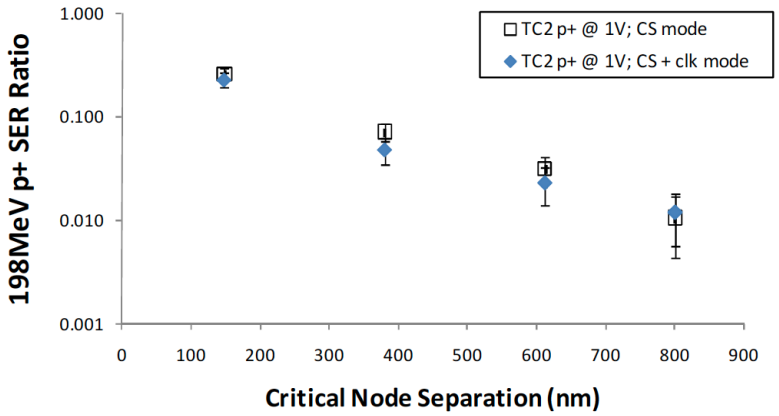


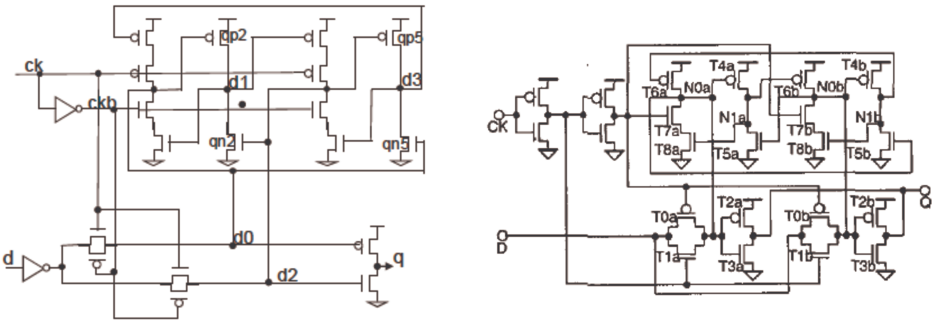
Figure 2.2: Effect of Node Separation on the SER of DICE Latch

state to be restored and as a result, a transient pulse may be seen at the output. When DICE is used to build a master-slave flip-flop, if an ionizing particle affects a transistor in the slave stage when it is opaque, even if the correct value is restored, a transient is visible at the Q output and can propagate downstream. Similarly, when the slave latch is transparent and the master is opaque, the glitch that comes out of the master will propagate to the flip-flop output. With increased operating frequency, there is less TDR and the effect of SETs induced in latches is becoming more significant [Alexandrescu 2013].

The pass transistors driven by the clock are not protected in the DICE cell thus upsets to these transistors can cause the stored value to be corrupted. Essentially an upset in the clock transistors can either (i) cause the cell to become transparent too early and latch the incorrect value or (ii) cause the cell output transition to be delayed causing a setup violation in the downstream stages [Seifert 2005]. In [Seifert 2010a], data is presented showing that without protection of clock nodes, the maximum reduction in SER that can be achieved with redundancy techniques does not exceed ≈30x.

2.2.2 SEUT

In the literature, especially in publications from Intel, a hardened cell called Single Event Upset Tolerant (SEUT) is frequently cited. Circuit diagrams for SEUT are found in [Hazucha 2003] and [Seifert 2010b] and reproduced in figure 2.3. The SEUT latch is very similar to the DICE latch in that four redundant storage nodes are configured in a feedback loop. The difference lies in the write logic. In the SEUT latch, two of the storage nodes have clock transistors in series in order to facilitate the write process. In terms of SER robustness, however, the SEUT is very similar to DICE.



(a) SEUT Latch (b) SEUT Flip-Flop

Figure 2.3: SEUT Latch and Flip-Flop

2.2.3 Graal

Graal [Nicolaidis 2007a, Nicolaidis 2007b, Yu 2009, Yu 2011] is a technique for protecting latches and is based on a two-phase, non-overlapping clock design style. The circuit structure for Graal is shown in figure 2.4. The odd numbered latches are clocked by $\phi 1$ and the even numbered latches are clocked by $\phi 2$. The duty cycle of both clocks is $\alpha : (1 - \alpha)$ and they are 180° out of phase. An XOR gate compares the D input and Q outputs of the latches and is used for timing fault, SET and SEU detection. The *error[i]* signals produced by the XOR gates from multiple latches are combined through an OR tree and then the combined error signal is captured on the falling edge of $\phi 1$ by an edge-triggered D flip-flop.

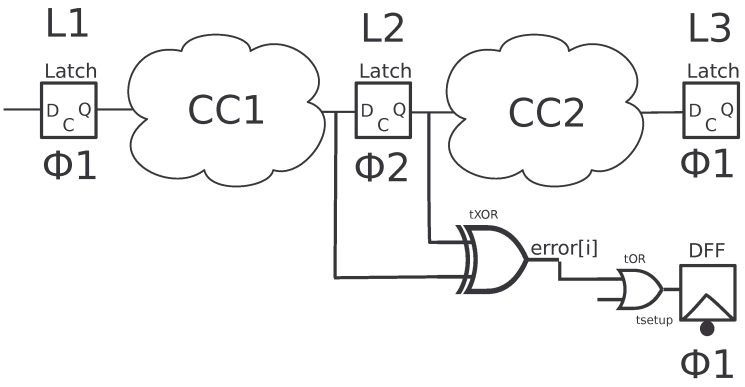


Figure 2.4: Circuit Structure of Graal

2.2.3.1 SEU Detection

In the Graal topology, SEUs may be detected by the XOR gates on the latches. The latch is only susceptible to SEUs when it is opaque (ϕ=0). Due to TDR in the logic

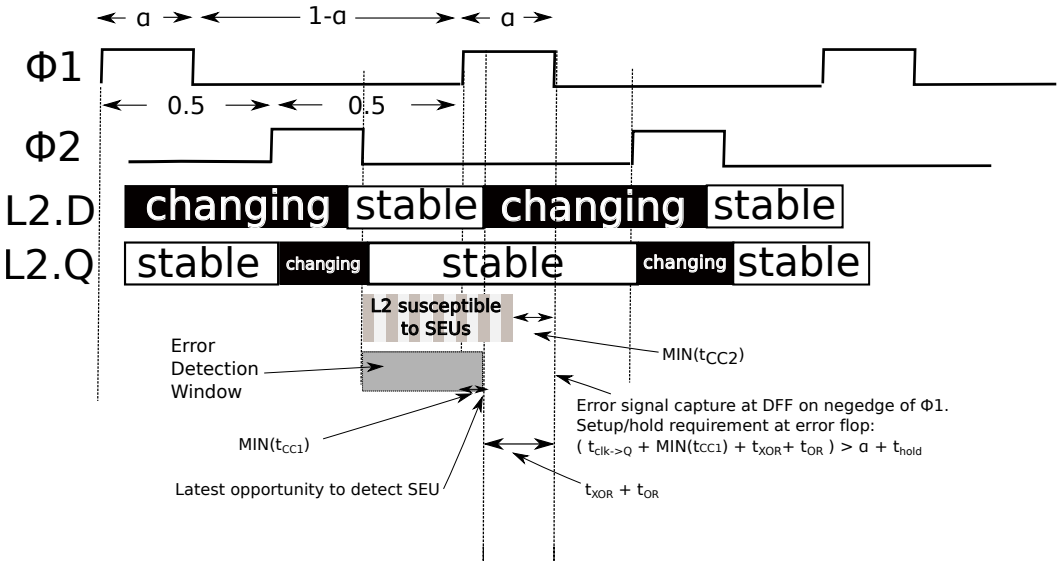


Figure 2.5: Graal Timing Diagram

downstream from the latch, the latest point in time at which an SEU can occur and affect the downstream logic is $(t_{clk} \cdot (1-\alpha) - MIN(t_{CC}))$ where $MIN(t_{CC})$ is the delay on the shortest combinatorial path. This region of susceptibility is labelled 'L2 susceptible to SEUs' in figure 2.5.

For an SEU to be detected by the XOR gate, there must be time for the error to propagate through the XOR gate, through the tree of OR gates and be sampled by the error flip-flop. Thus, the error must occur $(t_{XOR} + t_{OR} + t_{setup})$ before the falling edge of the clock. By ensuring that $(t_{XOR} + t_{OR} + t_{setup}) < MIN(t_{CC})$ full SEU protection is possible. This may be achieved through a combination of padding constraints on the logic (e.g. increasing $MIN(t_{CC})$) and a reduction the depth of the OR tree (e.g. minimizing t_{OR}).

2.2.3.2 SET Detection

Transient pulses due to SETs in the combinational logic can be detected by the XOR gate during the time interval from when the input of the latch is stable up to the point when the error capture flip-flop samples the error signal. Therefore, transients up to $(\frac{1}{2} \cdot t_{clock} - t_{XOR} - t_{OR} - t_{setup})$ in duration can be detected.

2.2.3.3 Timing Fault Detection

Similar to SETs, Graal can detect timing faults up to $(\frac{1}{2} \cdot t_{clock} - t_{XOR} - t_{OR} - t_{setup})$ in duration. Due to this ability to detect timing faults, Graal can be used for near threshold operation, assuming the circuit has a micro-architectural means to recover from the detected errors.

2.2.4 Biser

The Biser [Zhang 2006b, Mitra 2007] structure was proposed by researchers at Stanford and Intel. A Biser latch provides local correction of SEUs through the use of a redundant latch and a C-element which may include a weak keeper as shown in figure 2.6. A C-element is a small, asynchronous circuit where the output assumes the value of the inputs, only when both inputs have the same value.

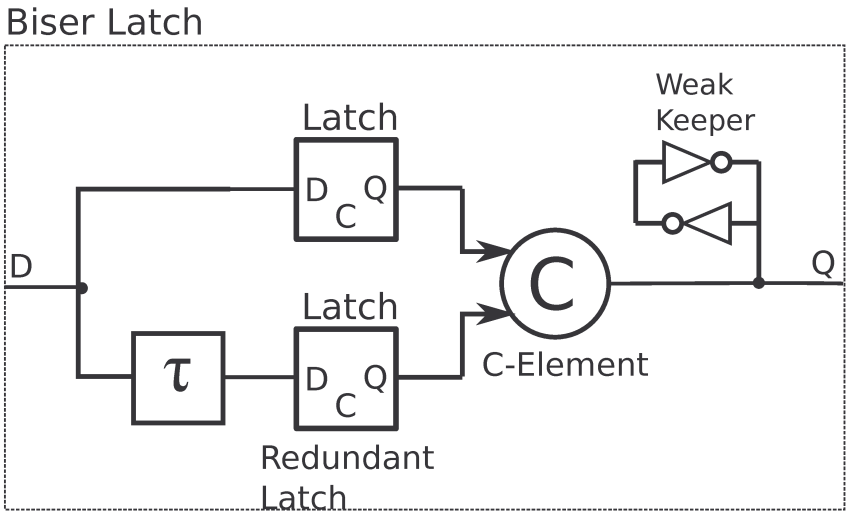


Figure 2.6: Circuit Structure of Biser

Variants of Biser have been proposed for latches and flip-flops, but in this chapter, we consider the latch version. In some design methodologies, redundant latches may already be present to support debug or test circuitry. In our comparison, however, we assume that this is not the case and account for the area overhead of the redundant latch.

2.2.4.1 SEU Robustness

Normally, the two latches sample the same value and this passes through the C-element. If a SEU affects one of the latches, then the upset value will not be propagated by the C-element which retains its old state. The C-element, however, increases the CLK $\rightarrow$ Q delay.

Of course with any hardened element based on redundant storage, the layout plays a critical role. In [Seifert 2010a] it is reported that SER sensitivity varies by nearly a factor of five when the minimum distance between sensitive nodes is increased by a factor of 2x as illustrated by the results reproduced in figure 2.7.

2.2.4.2 SET Robustness

The Biser structure is able to block SETs through the use of a delay element inserted on the input to the redundant latch. Transients are not a concern when the latches

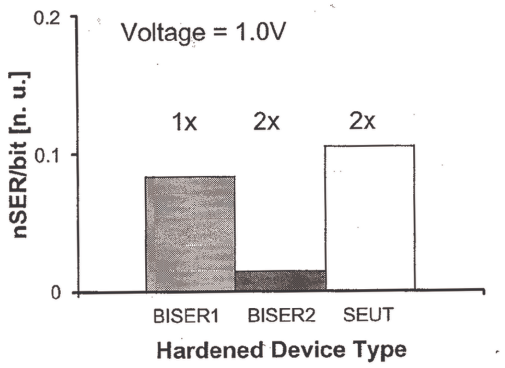


Figure 2.7: Impact of Layout on Biser Sensitivity

are opaque.

If a SET is present on the rising edge of the clock when the main latch becomes transparent, then there are two possibilities : either the transient was the same value as the old output value or it has the opposite value. In neither case does the transient propagate through the C-element. As long as the width of the transient is shorter than τ, then after this delay, both latches will see the correct value and only then will the output of the Biser latch be updated. After the correct value has been propagated, even while the clock remains high, transients of width less than τ are blocked by the C-element.

The longer the value of τ, the wider the transient that can be blocked, however this delay directly impacts the critical timing path, appearing as an increase in the *setup* time of the Biser latch.

2.2.4.3 Timing Fault Detection

The Biser topology is not able to protect against timing faults. Increased delay in the upstream logic simply passes through the latches when the clock becomes transparent, potentially causing timing violations in the downstream logic.

2.2.5 Razor-I

The Razor-I [Ernst 2004] flip-flop was presented by researchers at the University of Michigan. It is designed to *detect* timing faults and, coupled with DVS, enable significant power reductions[2]. The ability to detect timing errors allows the actual design to operate with zero timing margin and even with some negative timing margin, as long as the rate of occurrence of timing errors is sufficiently low.

The prototype design for Razor-I was an Alpha processor implemented in 180nm technology and only 192 of the total 2408 flip-flops actually used Razor flip-flops,

[2]The concept of double sampling with a delayed clock is covered by US Patent 7,380,192 (Nicolaidis) which was filed in March 2000 and pre-dates the Razor-I publications. The concepts used by Razor-I were published earlier in [Nicolaidis 1999].

as the maximum delay constraints on all the other paths could be met, even at the lowest operating voltage. A naïve approach to dealing with detected timing errors is to attempt to stall the entire processor. Obviously, this is not practical, as it would require routing signals from each point of error detection (e.g. 192 Razor flip-flops in the prototype) fanning out to every flip-flop in the full design (e.g. 2048 in the prototype). Instead, a counterflow approach was used in the original Razor-I prototype. With this technique, when a timing error is detected at a given pipeline stage, a bubble is propagated downstream and a re-try request is propagated upstream to the head of the pipe. This technique is relatively simple to implement, but it results in multiple processing cycles being lost after each error.

The basic Razor-I circuit is shown in figure 2.8. The main flip-flop is augmented with a shadow latch which is sampled with a delayed clock. An XOR gate compares the Q output of the flip-flop and the latch, generating an error signal. Due to the fact that the setup and hold constraints of the main flip-flop can be violated, a metastability detector is added to detect if the flip-flop has gone metastable. The output of the metastability detector is OR'ed with the result from the latch comparison (XOR gate) to provide an error signal. The error signals from many Razor-I flip-flops were combined using an OR tree to provide a local indication of a timing error. Internal to the Razor-I flip-flop, the error signal controls a local multiplexer which is used to restore the correct value to the input.

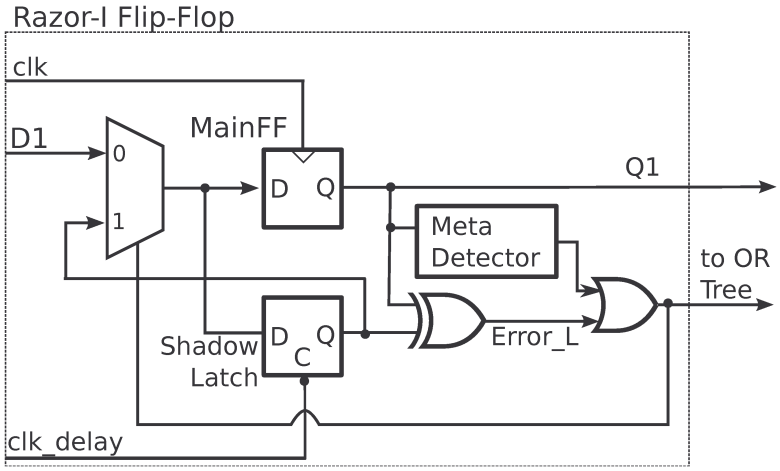


Figure 2.8: Circuit Structure of Razor-I

2.2.5.1 SEU Detection

It was not the original design intent of Razor-I to be robust to radiation effects, however, in order to compare its performance with other hardened sequentials, we investigate its SER robustness.

In the Razor-I topology, SEUs affecting the main flip-flop can be detected and corrected in the same way timing errors are treated. However, a SEU affecting the

shadow latch will *falsely* activate the correction mechanism causing the incorrect data in the shadow latch to be copied to the main flip-flop. Consequently, the Razor-I design does *not* provide protection against SEUs.

2.2.5.2 SET Detection

Depending on the duration of the transient pulse, the Razor-I technique may or may not detect the SET. The following situations are possible:

(a) If a SET is captured in the main flip-flop and the pulse is no longer present when the shadow latch samples its input, then the error will be detected and corrected in the same fashion as timing faults.

(b) A transient pulse that is longer than the delay between the sampling of the main flip-flop and the shadow latch can obviously not be detected.

(c) A SET that arrives after the sampling of the main flip-flop and that is sampled by the shadow latch will *falsely* trigger the correction mechanism. The incorrect value from the SET will be captured by the shadow latch and then transferred to the main flip-flop.

(d) A narrow transient pulse that occurs between the sampling of the main flip-flop and the sampling of the shadow latch will not be detected, however, such a transient is benign.

To summarize, the Razor-I design does *not* provide robust detection of SETs.

2.2.5.3 Timing Fault Detection

The Razor-I topology intrinsically detects and corrects timing errors that manifest themselves inside the monitoring window.

2.2.6 Razor-II

In an attempt to address the shortcomings of Razor-I, the team at the University of Michigan developed an improved version of Razor, called Razor-II [Blaauw 2008, Das 2009]. Because of the redundant latch, the area and power penalties of the Razor-I design are quite high. Furthermore, Razor-I provides no protection against radiation induced errors. The metastability detector, an integral part of Razor-I, is a difficult circuit to design correctly in the presence of process variation.

The Razor-II latch is based on a positive level-sensitive latch for the main storage. A simplified circuit diagram is shown in figure 2.9. The underlying design assumption is that the output of the latch should only transition shortly after the rising edge of the clock. A transition detection circuit is added to the output of the latch in order to detect timing faults, SEUs and SETs. To avoid false firing, the transition detector is disabled by a pulsed clock (DC) which goes low during a window from the rising edge of CLK through to the maximum CLK→Q delay of

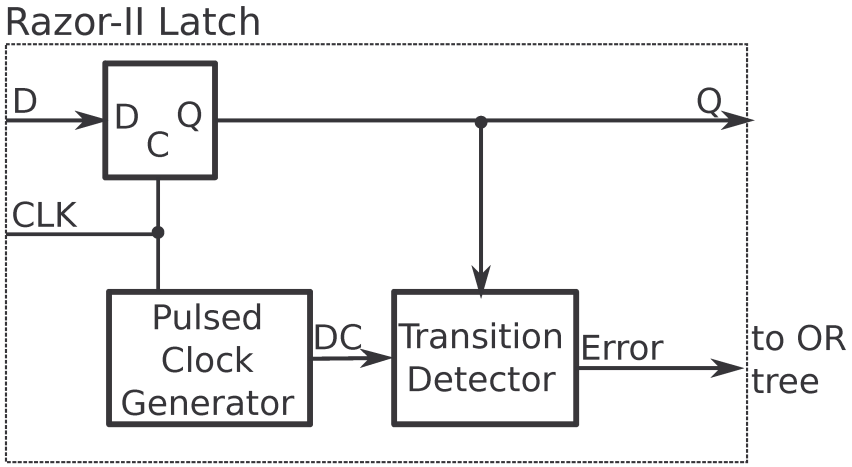


Figure 2.9: Circuit Structure of Razor-II

the latch. If a transition is detected when DC=1, it is assumed to be an error and is dealt with at the micro-architectural level. The Razor-II flip-flop is not able to perform any local error correction. A timing diagram is shown in figure 2.10.

In order to ensure that the D input to the latch is stable during the high period of the CLK, the duty cycle of the clock can be adjusted and short paths must be padded. If a clock with an asymmetric duty-cycle $\alpha : (1 - \alpha)$ is used, then all short paths must be padded so that their minimum delay exceeds $\alpha \cdot t_{clk}$ at all Process, Voltage, Temperature (PVT) corners.

2.2.6.1 SEU Detection

In the Razor-II topology, soft errors due to SEUs at the output of the latch are detected by the transition detector. If the latch output changes when DC=1, this is flagged as an error and recovered through micro-architectural techniques.

The low period of the DC clock must be designed such that it spans the worst case CLK→Q delay. If a SEU occurs during the low period of DC, it would be masked due the short path padding constraints imposed by this technique.

2.2.6.2 SET Detection

In case of SETs in the combinational logic, pulses that arrive at the input of the latch when it is transparent will be propagated to the output and detected by the transition detector. Of course, a SET that occurs when the latch is opaque is blocked by the latch.

2.2.6.3 Timing Fault Detection

Any timing fault which causes the latch output to transition after DC=1 will be detected and recovered through micro-architectural techniques.

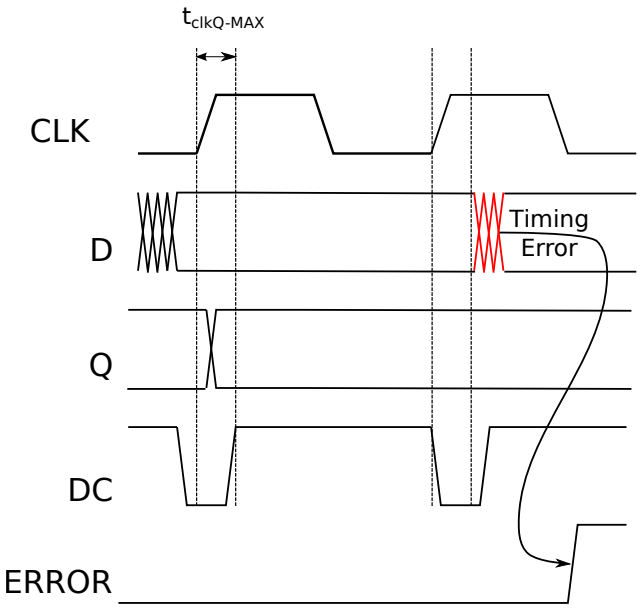


Figure 2.10: Razor-II Timing Diagram

2.2.7 TDTB and DSTB

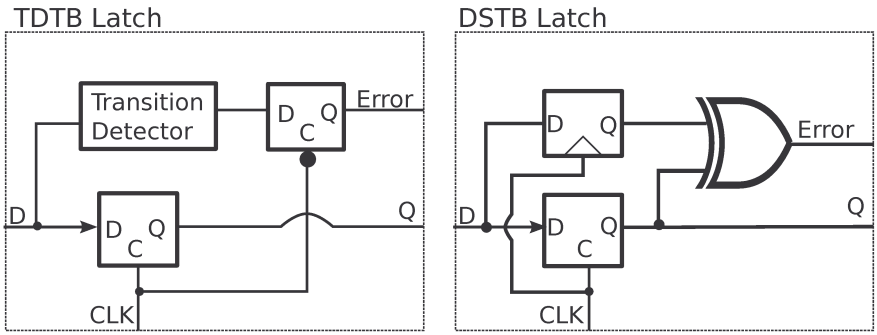
Two variants of double sampling flip-flops were investigated by Intel and described in [Bowman 2008]. The first, called Transition Detector with Time Borrowing (TDTB), is conceptually similar to the Razor-II design. It consists of a latch that is augmented with a transition detector as shown in figure 2.11(a). The transition detector flags any input transitions during the high-period of the clock, thus requiring the input to the latch be stable during the high phase of the clock. This can be achieved by adjusting the duty-cycle of the clock or by padding the short paths. This timing constraint is identical to that required for Razor-II (see section 2.2.6).

In Razor-I, the main storage element is an edge-triggered flip-flop and the shadow copy is stored in a latch. Douple Sampling with Time Borrowing (DSTB) is a variant of Razor-I where the main storage element is a pulsed latch and the shadow copy is captured using an edge-triggered flip-flop (see figure 2.11(b)). The SET and timing fault detection capabilities of TDTB and DSTB are similar.

In the published work, a small, three stage pipeline is implemented using both types of sequentials. It appears that the error signals from all the sequentials in the design are OR'ed together, fed to the first stage of the pipeline (Input Buffer) and caused the in-flight instruction to be re-issued with a half-speed clock. There are obvious scalability issues if the error signals fan into a single point in the design.

It is interesting to note, that the Razor-I design team required the use of a dedicated metastability detector in order to achieve robust operation [Das 2009] when the setup/hold constraints of the main flip-flop were not met. However, in the published work on DSTB and TDTB [Bowman 2008], metastability was apparently

not an issue, presumably because the main storage element is a latch.



(a) TDTB Latch (b) DSTB Latch

Figure 2.11: Circuit Diagrams of TDTB and DSTB Latches

2.2.7.1 SEU Detection

The difference between TDTB and Razor-II is that the transition detector in TDTB is only active during the high-phase of the clock. As a result, the TDTB does *not* detect SEUs.

The DSTB, however, does provide detection of SEUs. During the high phase of the clock, the main latch is not sensitive. During the low period of the clock, an upset in either the latch or the shadow flip-flop will trigger the error signal. Now, if the propagation time of the error signal to the control circuitry to initiate the recover ($t_{XOR} + t_{OR} + t_{setup}$) is longer than the propagation time of the shortest path on the main data-path, then the extent of the SEU detection will be reduced.

2.2.7.2 SET Detection

Both TDTB and DSTB provide detection of SETs that are narrower than the high phase of the clock. In TDTB, if the SET pulse overlaps with a portion of the clock high-period, it will trigger the transition detector. In DSTB, a SET shorter than the clock high period will be detected by the XOR gate.

2.2.7.3 Timing Fault Detection

Similar to SETs, TDTB and DSTB both provide detection of timing faults that are less than the clock high period in duration.

2.2.8 Bubble Razor

Building on the experience from Razor-I and Razor-II, the team at the University of Michigan developed the Bubble-Razor technique [Fojtik 2012, Fojtik 2013] for which they were granted US Patent *8,276,014* in September 2012. The starting

point of Bubble-Razor is a latch based design using two-phase clocking. The latches are augmented with XOR gates to detect errors and in this way it is similar to Graal (see section 2.2.3). The two-phase latch design style ensures that there is a temporal window when the inputs to each latch are stable.

Interestingly, the published work on Bubble-Razor emphasizes the system level error recovery rather than the circuit level error detection. Some details of the circuit implementation are presented in [Fojtik 2012] and an abstracted version of the basic circuit is shown in figure 2.12. The circuit is based on having a master latch that is augmented with a shadow latch. The master latch is transparent when the clock is high and drives the forward data-path. The shadow latch becomes opaque on the rising edge of the clock, thus keeping a sample of the input data. During the high-phase of the clock, the dynamic XOR gate compares the master and slave latches and if they mismatch, a timing error is reported. During the low phase of the clock, the dynamic XOR is pre-charged for the next cycle, and thus is not checking the outputs. The operation is shown in the timing diagram in figure 2.13. Note that the grey regions indicate when the latches are opaque.

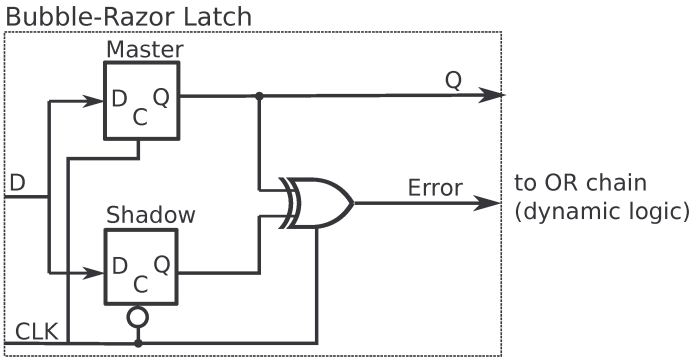


Figure 2.12: Circuit Structure of Bubble Razor

In the published papers [Fojtik 2012, Fojtik 2013], the major contribution of Bubble-Razor is the distributed approach for error recovery. The circuit is partitioned into *clusters*. Each cluster contains a group of latches clocked on the same clock phase. When a latch in one cluster detects an error, the error signal is propagated to the downstream clusters which are, of course clocked on the opposite phase of the clock. The propagation of this *bubble* causes the receiving clusters to gate their clock for one cycle, allowing extra time for the stage that had an error to complete its computation. The algorithm for propagating the *bubbles* through the design is carefully designed so that each cluster sees a *bubble* exactly once and ensures *bubbles* do not circulate infinitely around loops. If two different errors cause *bubbles* to be created, two waves of *bubbles* are created. A cluster receiving two incoming *bubble* requests only stalls once and only propagates a single *bubble*. In this way, the approach is able to handle multiple errors propagating through the system. Unlike instruction re-play and counter-flow techniques, this approach to

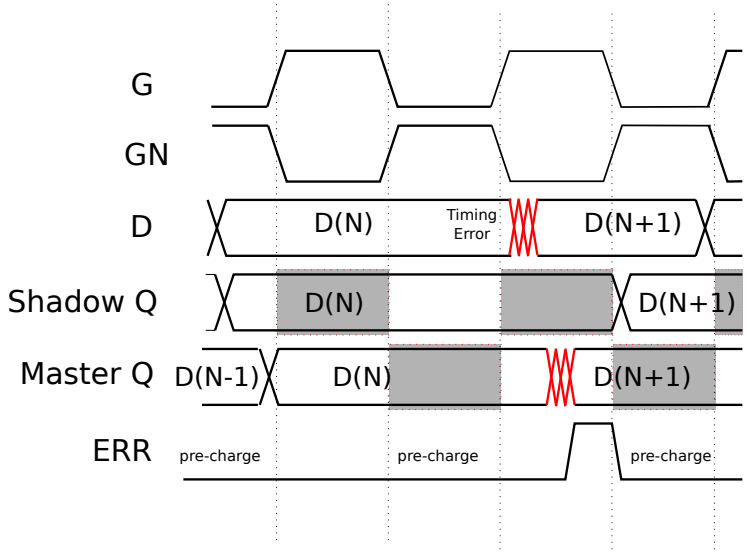


Figure 2.13: Bubble-Razor Timing Diagram

error recovery is not specific to a given design architecture (e.g. micro-processor pipelines, etc.).

The prototype design for Bubble-Razor was an ARM Cortex-M core implemented in a 45nm technology. The original flip-flop based design was converted automatically by a tool into a latch based design. On average each flip-flop was mapped into ≈3.29 latches and the area overhead for this transformation was 8%[3]. The overall area overhead for the Bubble-Razor implementation was 21%. For a typical process conditions, the Bubble-Razor techniques enables a performance increase of 22% at a given, fixed voltage. Alternatively, an energy reduction of 54% is possible at a fixed level of performance.

2.2.8.1 SEU Detection

The Bubble-Razor technique does not provide protection against SEUs. The XOR gate which compares the outputs is only active when the master latch is transparent and during this time, the latch is not susceptible to SEUs. During the time when the clock is low ($G = 0$), the dynamic XOR gate is being pre-charged and thus does not detect errors.

2.2.8.2 SET Detection

The Bubble-Razor technique can detect SETs. If a SET is present on the rising edge of the clock, the incorrect value will be sampled in the shadow latch. Later, when the transient has expired, the output of the master latch will assume the correct

[3]Ideally each flip-flop would map to two latches. In practice, when the flip-flop circuit is retimed, the cut-point can be wider and additional latches are required.

value and the error detection logic will be triggered. The maximum width of SET than can be detected is equal to the high-phase of the clock less the propagation time for the error signals.

2.2.8.3 Timing Fault Detection

Bubble-Razor does effectively detect timing faults. The maximum timing fault that can be detected, is limited by the fraction of the cycle the clock is high, less the propagation time of the error signal through an OR tree.

2.2.9 EDC Flip-Flop

The Error Detection Correction (EDC) flip-flop [Valadimas 2010] consists of a main flip-flop, an XOR gate, a redundant latch and a multiplexer. The XOR gate compares the input and the output of the main flip-flop and this result is sampled by the latch as shown in figure 2.14. The idea is that the data at the input to the flip-flop must remain stable for a period of ϕ so that the main flip-flop and the redundant latch sample the same value.

The latch is clocked with a locally generated pulsed clock that is delayed from the main clock by a delay ϕ as shown in figure 2.14. The width of the pulsed clock is α and it must be large enough to meet the minimum pulse-width requirements of the latch, under all operating conditions. The logic to generate the pulsed clock can be shared amongst a group of flip-flops. Increasing the value of ϕ is costly both in terms of area, due to the additional delay elements, but more importantly in terms of active power because the delay gates on the clock generation circuitry are toggling at the full clock rate.

If a timing error is detected, then the *Error* signal causes the clock to be globally gated to the entire circuit for one cycle. During this time, the inverted (Qbar) output of the main flip-flop is output and thus the correct computation occurs. This approach imposes a tight timing constraint: in less than one cycle, the *Error* signal must be generated at every EDC flip-flop and then propagated to the clock input of every flip-flop in order to perform the clock gating. This constraint limits the operating frequency and the size of the circuit.

2.2.9.1 SEU Detection

A SEU that occurs in the main flip-flop between the rising edge of the clock and the falling edge of *C_Pulse* will be detected by the XOR gate. A SEU that occurs later in the clock cycle will go undetected. Therefore the overall fraction of SEUs that are detected is $\frac{\phi+\alpha}{t_{clock}}$. In practice, due to the requirement that all combinatorial paths be padded to a minimum delay of ϕ and due to the power cost in generating the delayed clock, this fraction will be small and overall the EDC flip-flop provides only limited SEU protection.

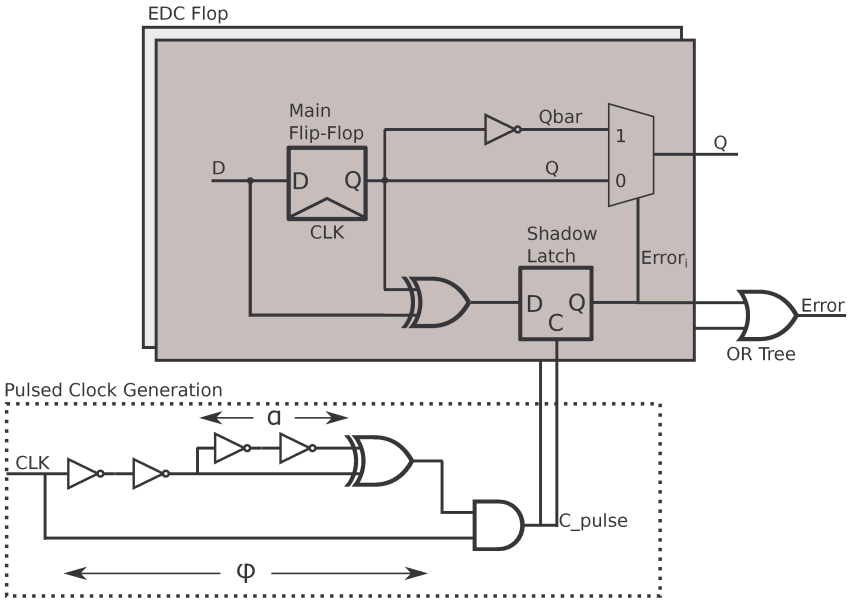


Figure 2.14: Circuit Structure of EDC Flip-Flop

2.2.9.2 SET Detection

A SET that is shorter in duration than $\alpha + \phi - t_{setup}$ will be safely detected. Any SETs that are longer, will go undetected because the redundant latch will sample the incorrect value.

2.2.9.3 Timing Fault Detection

Similarly to SETs, a timing fault up to $\alpha + \phi - t_{setup}$ will be detected.

2.2.10 SET Flip-Flop

The *SET* or Soft Error Tolerant flip-flop [Valadimas 2012] is designed to correct SEUs and it draws upon aspects of the EDC flip-flop and aspects of Razor-II. The circuit is shown in figure 2.15. The idea is that the flip-flop detects unexpected transitions at the output and it corrects them using the preset or clear inputs to the flip-flop. Normally, the output of a flip-flop only transitions during a short window of time after the active clock edge. Any transitions on Q that occur after the maximum CLK $\rightarrow$ Q delay must be caused by an SEU.

The SET flip-flop uses a Transition Detector (TD) to detect an unexpected output transition. To prevent the TD from firing spuriously, an enable signal, *TDE*, is used to gate it off for a short period after the clock edge. The *TDE* signal is normally high, but pulses low for a period at least as long as the maximum CLK $\rightarrow$ Q delay. If an unexpected transition is detected, two actions occur. First, the error is latched in an Error Flip Flop. In addition, the correct value is restored in the main flip-flop by activating either the *preset* or *clear* inputs, as shown in

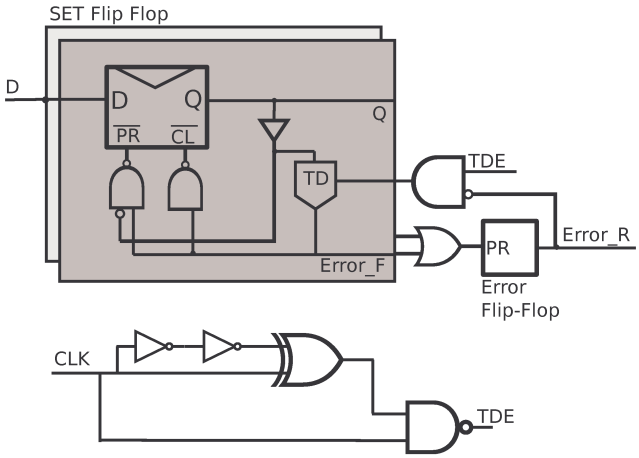


Figure 2.15: Circuit Structure of SET Flip-Flop

figure 2.15. Care is required to ensure that this corrective action does not re-trigger the TD creating an oscillation. This is prevented by disabling the correction path once the error flip-flop is activated.

The area overhead and design complexity of the SET flip-flop are quite significant. A simplified version of the design is shown in figure 2.16. This variant provides only detection and the TD is shared between two flip-flops by using an XOR gate. This variant is attractive in cases where micro-architectural correction techniques can be used.

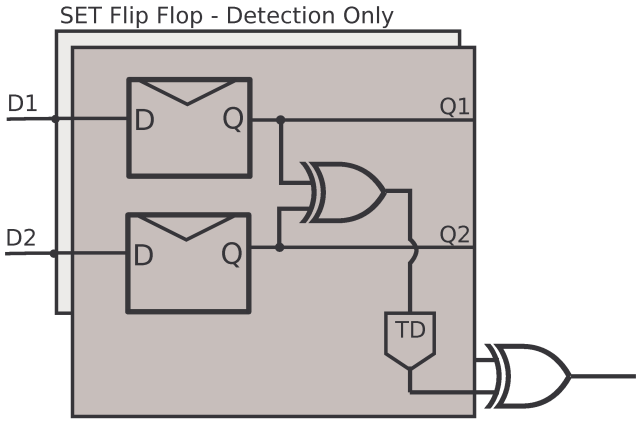


Figure 2.16: SET Flip-Flop with Detection Only

2.2.10.1 SEU Detection

Generally, the *SET* flip-flop provides reasonable SEU protection. There is, however, a small window of time shortly after the clock edge, when an SEU may not be detected. This is because the enable for the timing detector (TDE signal) must be

deisgned to cover the *worst case* CLK → Q delay which may be significantly longer than the actual delay.

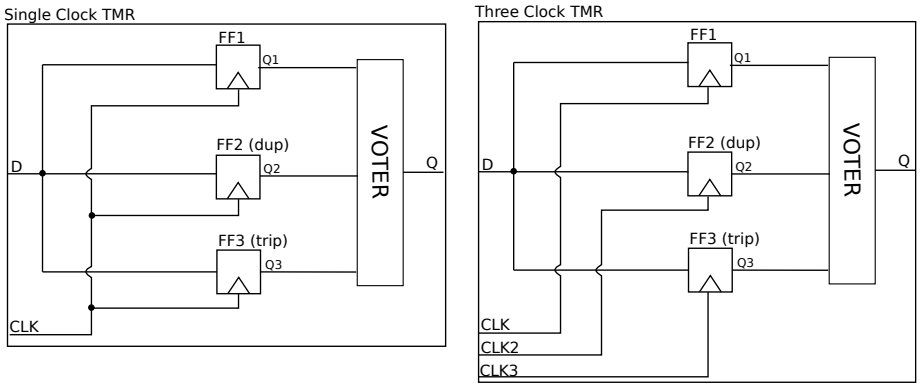
When an SEU occurs, during the window of time it takes for the correction to occur, the incorrect value will propagate. The intended use of the SET flip-flop is that when an error is detected, the clock is gated, and extra time is allowed for correct computation to occur using the corrected value.

2.2.10.2 SET and Timing Fault Detection

The SET flip-flop does not provide a mechanism for detecting or correcting SETs or timing faults.

2.2.11 TMR

When the highest level of robustness is required, Triple Modular Redundancy (TMR) techniques are typically used. In the most common form of TMR, at the circuit level, just the flip-flops are triplicated as shown in figure 2.17(a). This approach only provides protection against SEUs and is often used when hardening Field Programmable Gate Array (FPGA) designs.



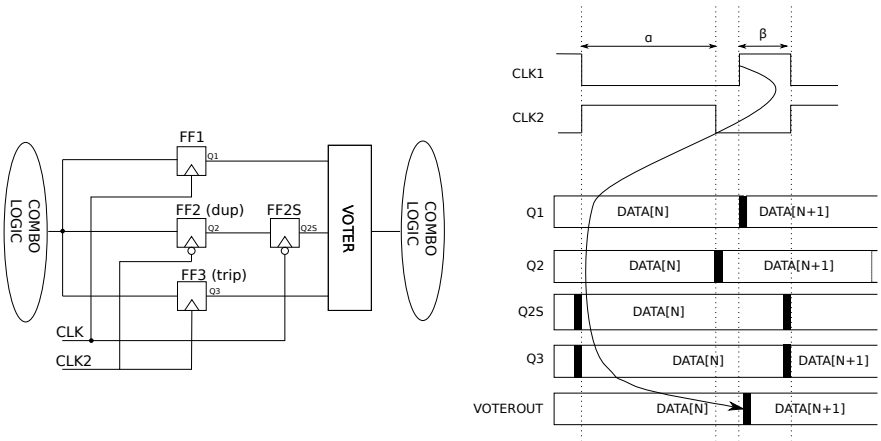
(a) Single Clock TMR (b) Triple Clock TMR

Figure 2.17: Use of TMR For Flip-Flop Protection

Since the clock network is not replicated, it remains a weak point, and any transients on the clock will affect all the flip-flops. In space applications, this is addressed by replicating the clocks as shown in figure 2.17(b). If the phase difference on the replicated clocks is at least t_{SET}, then any transients of less than this duration will be filtered as the transient is then only sampled by a single flip-flop. Of course, this comes at a penalty in timing performance equal to twice the transient pulse width.

Having three clock distribution networks is very costly and recently techniques to achieve the same robustness using only two clocks have been developed as shown

in figure 2.18 [Nicolaidis 2013]. With this approach, the third temporal sampling point is obtained by adjusting the duty-cycle of the clocks.



(a) TMR Technique with Two Clocks (b) Timing Diagram for TMR with Two Clocks

Figure 2.18: Flip-Flop TMR Using Two Clocks

2.2.12 Parity Codes

Parity codes have been used to protect data since the earliest computers. In some situations, they can be very effective at protecting flip-flops against SEUs. In figure 2.19, it is shown how parity can be calculated over a set of data bits and the parity stored in an additional flip-flop. On the output of the flip-flops, the parity is checked and errors are detected. The major drawback to parity codes is that the additional delay to compute the parity directly affects the critical path.

For parity generation, the cost is half an XOR gate, per bit. On the checking side, the parity must be re-computed, which also costs half an XOR gate per bit plus a final XOR to compare the result with the stored parity. The cost of the extra flip-flop for the parity, is amortized across the N flip-flops that are being protected. To combine the error indications from multiple groups of parity bits, an OR tree is required which is an additional overhead.

Parity codes can, however, be very effective when data moves through several stages of flip-flops without being modified. In this case, the parity can be calculated at the front of the pipe and checked at the end. The overhead of the parity calculation and checking circuitry is amortized over the M stages, as shown in figure 2.20. Only an extra parity bit needs to accompany the data bits through the intermediate stages. In this case, per-bit the area overhead is approximately $\frac{A_{XOR}}{M} + \frac{A_{FF}}{N}$.

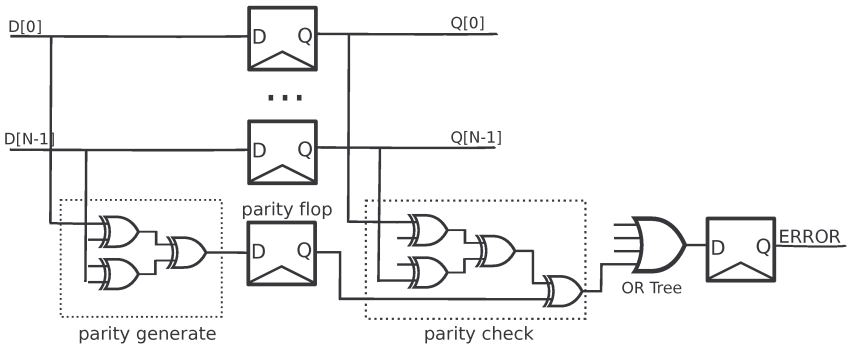


Figure 2.19: Flip-Flop Protection Using Parity

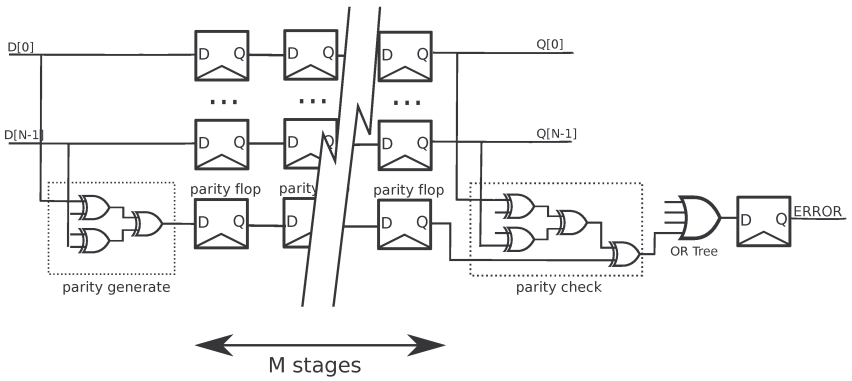


Figure 2.20: Protection of a Path Using Parity

2.2.12.1 SEU Detection

The basic parity technique provides strong robustness against SEUs. Using placement constraints to ensure that the flip-flops protected by the same parity bit are not physically adjacent, the technique can also provide strong protection against Multiple Bit Upsets (MBUs). This is interesting to note, as techniques such as DICE do not provide protection against upsets to multiple nodes. Improving the layout of a DICE cell to provide robustness against strikes to multiple nodes can have an area overhead of 40% [Lee 2010]. In a fast technology, where the timing penalty may be acceptable, parity remains an attractive technique.

2.2.12.2 SET and Timing Fault Detection

The parity structure shown in figure 2.19 is not intended to provide detection of SETs and timing faults. However, any SET that is shorter in duration than the delay through the parity logic will actually be detected.

There is a significant timing penalty to insert the parity generation logic. This extra delay can be viewed as a form of timing margin that is imposed on the circuit. If timing violations do occur, they will occur first on the parity path, before they

appear on the data-path. One could thus say that parity logic does provide detection of timing faults, although this comes at the cost of a significant timing overhead.

2.3 Comparison of Robust Sequentials

As seen in the above sections, there exist a large number of robust sequential cells, each with specific error detection capabilities, different overheads and specific usage constraints. In this section, the objective is to compare the different techniques and put them in context.

It is often difficult to assess the implementation cost of a cell without completing the transistor-level design. The overheads are technology specific and even for a given type of cell, multiple implementations with different drive strengths and speeds are possible. However, it is clear that certain circuit designs have intrinsically higher overhead than others. The approach taken to compare the area of the different circuits is based on counting the number of additional transistors required to implement the robust cell, compared to a minimum-sized implementation of either a standard latch or flip-flop. It is true that transistor widths vary greatly, so this measure has limitations, however, to correctly size the transistors requires a full implementation of each design, which is beyond the scope of this thesis.

The Nangate Open Cell Library [Nangate 2008] was used to facilitate the comparison. The transistor count and area of different standard cells is shown in table 2.1. The library does not include a C-element, but from a schematic it is known that a C-element can be implemented with 12 transistors.

Table 2.1: Transistor Counts for Basic Design Elements

Description	**Cell**	**Transistor Count**
INV gate	INV_X1	2
BUF gate	BUF_X1	4
OR gate	OR2_X1	6
XOR gate	XOR2_X1	10
NAND2 gate	NAND2_X1	4
2-input mux gate	MUX2_X1	12
Latch	DLH_X1	16
D Flip-Flop	DFF_X1	28
D Flip-Flop w SET,CLR	SDFFRS_X1	52
C-element		12

For some of the robust sequentials, there is extra circuitry that can be shared across multiple flip-flops. Specifically, this is the case of the pulsed clock generators used in the EDC and the *SET* flip-flop. Also, for those designs that provide detection, there is an OR tree which combines the error signals and terminates at an error flip-flop. For the purposes of this comparison, it is assumed that these structures are shared amongst 32 flip-flops. In this way, a per bit overhead is calculated in

terms of the number of additional transistors. Using this approach, the area and timing overheads for the robust sequentials studied in this chapter are summarized in table 2.2.

Table 2.2: Comparison of Hardened Sequentials

Circuit	Timing Impact				Area	Detection			Correction		
	setup	hold	CLK→Q	Padding		SEU	SET	TF	SEU	SET	TF
Single Phase Latches											
DICE	≈0[e]	≈0	≈0	No	10T				✓		
Biser	t_{SET}	0	t_{Celem}	No	36T				✓	✓	
Razor-II	0	0	0	Yes	31T[b]	✓[f]	✓	✓			
TDTB	≈0	0	0	Yes	30T		✓	✓			
DSTB	≈0	0	0	Yes	38T	✓[a]	✓	✓			
Two Phase Latches											
Graal	≈0	≈0	≈0	Yes	14T	✓[a]	✓	✓			
B-Razor	≈0	≈0	≈0	No	30T[d]		✓	✓			
Flip-Flops											
Razor-I	t_{MUX}	0	0	Yes	48T[c]			✓			✓
EDC	0	t_{SET}	t_{mux}	Yes	44T		✓	✓		✓	✓
parity	$N \cdot t_{XOR}$	0	0	No	12T	✓					
SET FF	0	0	≈0	No	53T	✓			✓		
SET FF det	0	0	≈0	No	21T	✓					

[a] The extent of SEU detection depends on timing constraints. See section 2.2.3.2 or section 2.2.7.1 for details.

[b] Reported transistor count of Razor-II is 47. Basic latch assumed to have 16 transistors. Overhead is thus 31=47-16.

[c] Reported transistor count of Razor-I is 76. Basic flip-flop assumed to have 28 transistors. Overhead is thus 48=76-28.

[d] Overhead for bubble-Razor consists of a shadow latch (16), a dynamic XOR gate (10) and at least one dynamic OR gate (6).

[e] In [Hazucha 2003] timing overheads for a DICE implementation are presented and are under 2%.

[f] The SEU detection of Razor-II is limited by the fact that the low period of the DC clock must span the worst case CLK→Q delay. See section 2.2.6.1.

2.4 Conclusions

To provide a simple summary of the protection capabilities of the different techniques, the cells have been grouped into three sets based on whether they mitigate SEUs, SETs or timing faults. A Venn Diagram showing these capabilities is shown in figure 2.21. Cells which provide only detection are shown with a dark, red square and those that provide correction are shown with a light, blue square.

For SEU robustness alone, the DICE cell is well established, including variants with improved layout (e.g. LEAP-DICE [Lee 2010]). However, it is important to remember, as pointed out in section 2.2.1.1, when an upset occurs in a DICE cell, it takes some time to recover and a glitch can propagate into the downstream logic. At higher frequencies, there is less TDR and thus an increased risk that such a glitch will be sampled downstream.

When looking for protection against both SEUs and short SETs, the Biser cell

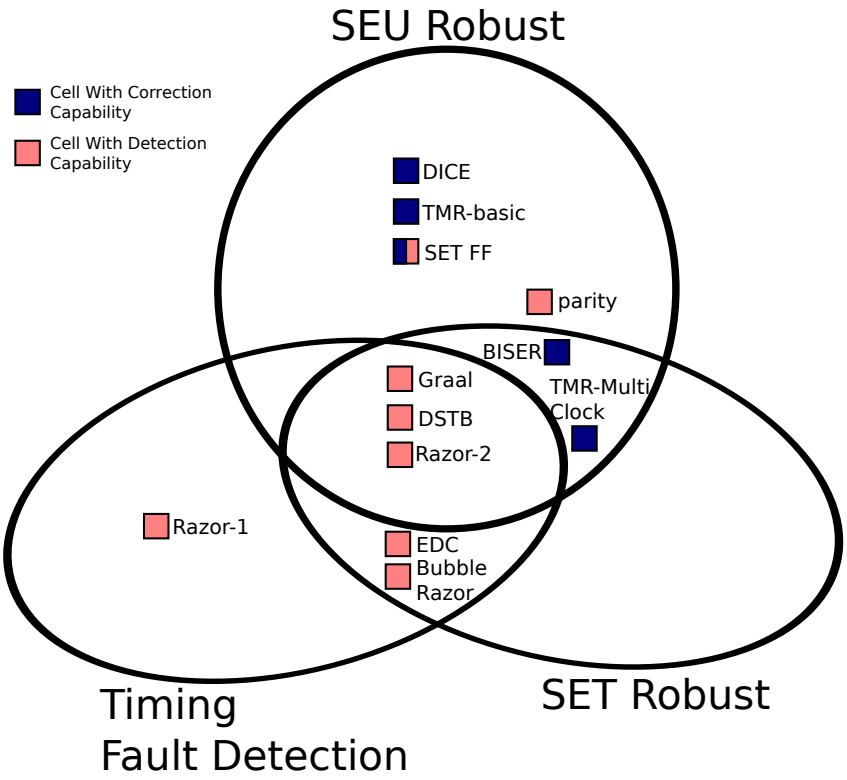


Figure 2.21: Venn Diagram Showing Properties of Robust Sequentials

is attractive. For a cost slightly higher than a redundant latch, through the use of a C-element, the Biser can block SEUs and SETs. In order to detect wider SETs, the length of the delay chain (τ delay) at the front of the cell must be increased.

We identify three cells (Graal, DSTB and Razor-2) which, in principle provide detection of SEUs, SETs and timing faults. All three are latch based designs and have special timing constraints. Using two-phase, non-overlapping clocking, Graal ensures that the inputs to each latch remain stable after the latch is opaque. To achieve SEU protection during the full window of temporal susceptibility, either the short paths must be padded or the depth of the OR tree must be very shallow (see section 2.2.3.1). Razor-2 achieves SEU detection using a transition detector that is disabled around the active edge of the clock. The disabling of the transition detector creates a window when SEUs may not be detected. Razor-2 imposes the requirement that short paths be padded to ensure that the inputs to the latches are stable during high period of the clock. The timing requirements for DSTB are similar to those of Razor-2, thus requiring short paths to be padded.

Although designs exist that can detect all three classes of errors, they impose specific timing constraints and have some limitations on the extent of SEU detection.

Chapter 3

Single Event Effect Analysis

Contents

3.1 Overview of Analysis Techniques

Analyzing the effect of faults induced by soft errors in complex integrated circuits remains challenging. The vast majority of faults do not propagate due to the various de-rating factors that were identified in section 1.3. Without an accurate characterization of the fault propagation probabilities, the failure rate of the system may be grossly misestimated. Usually the raw soft error rate of basic cells (e.g. memory cells, flip-flops) can be characterized accurately and error bars of under 10%-15%

are typical. Achieving the same level of accuracy in the effects analysis is difficult and often the accuracy of the full system-level analysis is limited by the accuracy of the effects analysis.

A high-level view of the propagation from faults to system level failures is shown in figure 3.1. Shown on the left are the faults that can be induced in flip-flops, logic gates and memories. Due to de-rating effects, Temporal De-Rating (TDR), Logical De-Rating (LDR), Electrical De-Rating (EDR) and Memory De-Rating (MDR), many of these faults do not propagate. After applying these de-ratings, the effective rate of visible errors is calculated. The next step is to analyze the effect of each error on the system. This comes from the FDR analysis and the result is the rate of occurrence for a defined set of system-level failures.

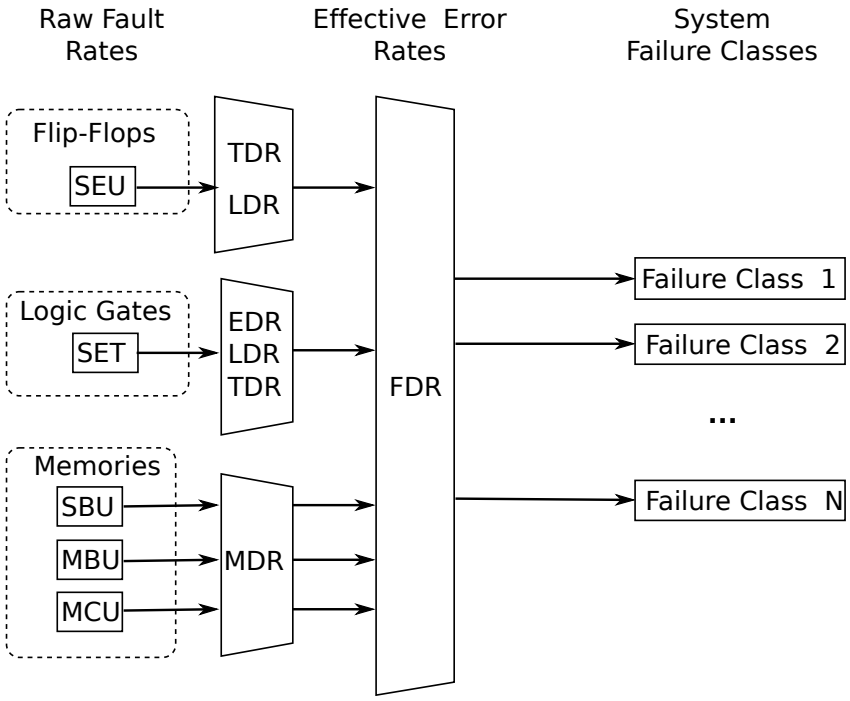


Figure 3.1: Soft Error Effects Analysis

The most common approaches to evaluate fault propagation centre around fault-injection analysis. Many faults are injected into a model of the circuit and the effect is observed. Depending on the accuracy of the model (e.g. gate-level, RTL, etc.) only some of the de-rating factors may be taken into account. There exist many different fault injection techniques and a brief review is presented in section 3.1.2. Often FPGA-based platforms are used to accelerate fault-injection and these are discussed in 3.1.2.2.

Analytical approaches are an alternative to fault injection. These approaches use a mathematical or abstracted model of the circuit to evaluate the effect of faults. Microprocessors have been studied in depth and a widely used analytical approach is based on the notions of Architectural Vulnerability Factor (AVF) and Program

Vulnerability Factor (PVF) and these are discussed in section 3.1.3.

Analytical techniques also exist for circuit level fault effect analysis. Some of these use special data-structures such as Binary Decision Diagrams (BDDs) or Algebraic Decision Diagrams (ADDs) to represent the circuit and reason about fault propagation. Other techniques are based on a formal representation of the circuit as well as the properties it must meet. Using tools for model checking or symbolic simulation, it can be verified whether the properties hold in the presence of faults. These techniques are discussed in section 3.1.4 and 3.1.5 respectively. Another set of approaches is based on associating probability distributions to the nodes in a circuit and then using these to compute the probability that faults on a given node will propagate to an output. These techniques are reviewed in section 3.1.6.

We conclude the first half of the chapter with a qualitative comparison of the different techniques in terms of their accuracy, speed, the types of faults that can be modeled and the types of masking that can be analyzed. In the second half of the chapter (section 3.2), a detailed case study of the soft error effects for several large design blocks in a Network Processor (NP) is presented.

In practice, no single technique can be applied to an entire System on Chip (SoC). Instead hierarchical and hybrid approaches must be developed. In chapter 5, a hierarchical technique for analyzing the effects of SETs is presented. The RIIF modeling language which is described in chapter 6, is intended to facilitate such hierarchical analysis by providing a means to encapsulate the results of a block-level analysis and make them available for an analysis at the next higher level of hierarchy.

3.1.1 Objectives of Soft Error Effects Analysis

The goal of any soft error effects analysis is to assess the probability that radiation induced faults result in the occurrence of a chip or system-level error. The list of chip or system-level failures must be clearly enumerated prior to the analysis. In processor applications, the two main effects that are typically identified are Silent Data Corruption (SDC) and Detected Uncorrected Errors (DUEs) [Mukherjee 2008]. A common taxonomy for processor effects is reproduced from [Weaver 2004] in figure 3.2. SDC occurs when a bit has no protection and its value affects the program execution. DUE occurs when there is a fault in a bit which has only detection, regardless of whether it affects the program outcome, or not. A well known consequence of the latter observation is that by simply adding error detection to a processor, the SDC rate may be reduced, but this comes at the expense of increased DUE. Leading processor manufacturers characterize the SDC and DUE rates of their designs and communicate these values to their customers, usually under strict Non-Disclosure Agreements (NDAs).

In core networking applications (routers and switches), a different approach to classifying the effects of faults is proposed in [Silburt 2008, Silburt 2009]. The specification that is presented defines outage classes based on the duration of time the network is unavailable. A very short outage, affecting only a small number of pack-

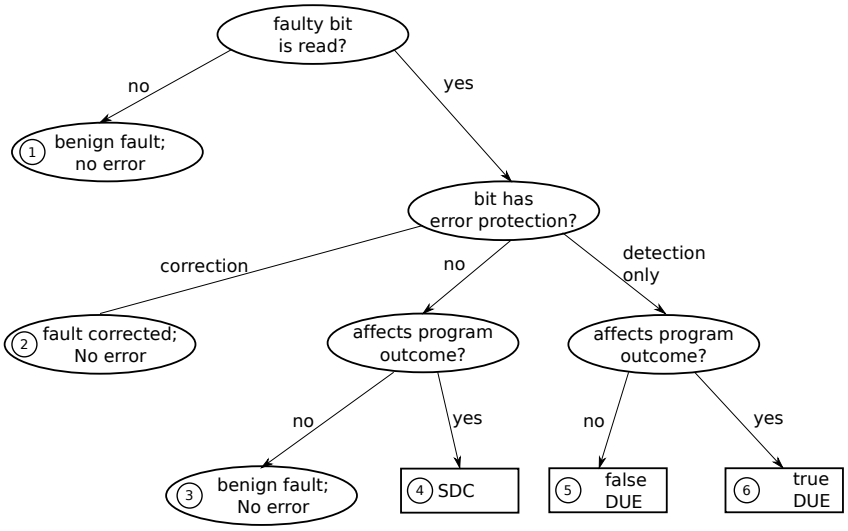


Figure 3.2: Classification of Faults in Processors

ets, may go undetected, even by the neighbouring network elements. Such low severity outages can occur relatively frequently without impacting the network operation. As the duration of the outage increases, so does the severity. Using Quality of Service (QoS) specifications for packet traffic, card-level hardware reliability targets and network availability requirements (e.g. five 9's), the authors establish bounds on the rate of occurrence of different classes of system failures and these are summarized in table 3.1. Defects per Million (DPM) is an availability metric.

Table 3.1: Outage Classes and FIT Rate Targets for Internet Core Routers

Class	**Maximum Outage Time (s)**	**Maximum Outage Rate (FITx1000)**	**DPM**	**Maximum Down Time (s/year)**
No outage	0	Unlimited	0	N/A
Low	0.01	30	0.000	0.00
Medium	3	10	0.008	0.26
High	180	1	0.050	1.58
Unrecoverable	1000	1	0.278	8.77
		Total	0.336	10.61

When analyzing an ASIC used in networking applications, it is necessary to ensure that these system level targets are met for each outage class. As part of the FDR analysis, each fault must be classified based on the class of outage it produces. Then the total failure rate for each outage class is summed for the entire system. To summarize, the goal of a soft error effects analysis is to compute the rate of system level failures starting from the rate of technology level faults due to soft errors.

3.1.2 Fault Injection

Fault injection analysis consists of injecting a number of faults into a model of a system to evaluate how it behaves in the presence of these faults. Models with different abstraction levels can be used. Obviously, there is a trade-off between the level of abstraction versus the speed of the analysis and the accuracy of the results. The main challenge with any fault injection analysis is managing the number of faults that must be evaluated. This number grows with:

(i) Number of fault types (e.g. SEUs, SETs).

(ii) Design size (e.g. number of gates).

(iii) Number of workloads or stimulus patterns to analyze.

(iv) Duration in cycles of each workload or stimulus pattern.

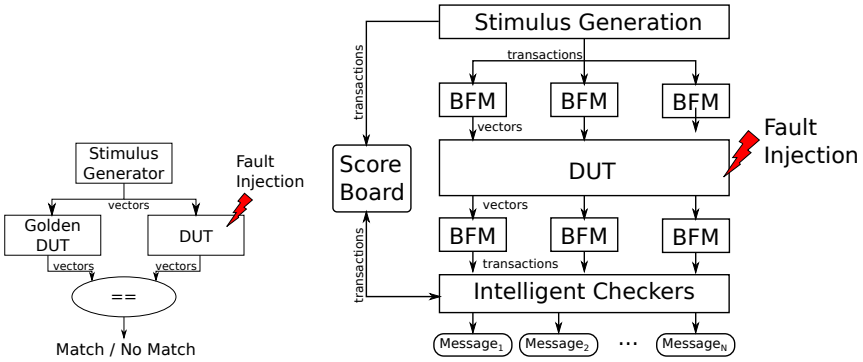
The review presented in [Yu 2005] exposes the fact that exhaustive fault simulation is not possible for commercial circuits. For example, in [Valderas 2010] an exhaustive fault injection analysis was performed on a small micro-processor with 596 flip-flops and required 80 million fault injections. Many techniques to accelerate fault-injection simulations have been proposed and are discussed in section 3.1.2.1. An enhancement to existing acceleration techniques is presented as part of the case study in section 3.2 of this chapter.

When the objective of the analysis is to compute system-level failure rates, it is only necessary to evaluate a relatively small number of faults in order to obtain accurate estimates of the system level error rates. In section 3.3, error bounds for statistical fault injection are discussed. When the objective is fault grading (e.g. identification of which circuit elements are most critical), new techniques are required to identify the critical elements in large designs and this is the topic of chapter 4.

3.1.2.1 Simulation Based Fault Injection

Independent of reliability concerns, digital simulation is used extensively to verify the functional correctness of modern designs. The environment used for functional simulation can be reused for fault simulation. A simple approach for simulating faults is shown in figure 3.3(a). With this approach, the Device Under Test (DUT) is instantiated twice. A fault is injected in one instance while the other serves as a 'golden' reference and the output vectors from both instances are compared clock by clock. In practice, as long as the input vector sequence remains fixed, it is not necessary to actually simulate the golden version of the DUT during each run as the output vectors can simply be recorded in advance. It is, however, only possible to assess whether the output vectors diverge. This approach provides no insight into the actual effect of the divergence.

The advantage of this simple approach is that it can be applied to almost any design and does not require an understanding of the application. The corollary is that this approach provides no insight into FDR.



(a) Simple Simulation Environment

(b) High Level Simulation Environment

Figure 3.3: Fault Injection Simulation Environments

Figure 3.3(b) shows a more complete view of a modern functional simulation environment. The stimulus is generated in terms of transactions (e.g. packets, video frames, numeric data, etc.) which are converted to low-level vectors by Bus Functional Models (BFMs). On the output side, the raw vectors from the DUT are parsed into transactions which are checked by intelligent checkers. The stimulus generator and the checker communicate via a scoreboard which is a data-structure that tracks the transactions currently being processed by the DUT. As the intelligent checker works at a high level of abstraction, it detects and reports errors at the transaction level (e.g. missing packets, corrupted video frames, numerical errors, etc). Typically, these errors are written to a log file and it is possible to map the various error messages to the system-level impact they have. Essentially, the intelligent checkers provide an automated means to evaluate the system level effect of faults.

The challenge with simulation based fault injection is the relatively low speed of digital simulators. Fortunately, the problem of fault-injection is easily parallelized. By the 1970's [Chang 1974] techniques for parallel fault simulation had been developed and already at this time, it was possible to evaluate thousands of faults per second on mainframe computers for circuits with hundreds of gates. Below, we discuss a few of the numerous techniques that have been developed to accelerate fault simulations.

Recently, [Alexandrescu 2012] presented the implementation of an acceleration technique based on exploiting the fact that modern processors typically have a 64-bit word width. On the host processor, instead of running simulations sequentially, up to 64 simultaneous fault injections can be performed, where each bit in the data-path of the host processor is used to evaluate a different scenario. A tool was developed

which can compile an RTL description into C code which, when executed, evaluates 64 faults in parallel. The combination of the parallel simulation and the optimized C code, results in a performance increase of over 500X compared to a commercial event based simulator. The downside is that it can only be used to implement simple fault injection, as shown in figure 3.3(a) and it does not provide a means to parallelize the intelligent testbench bench components shown in figure 3.3(b), thus limiting its applicability to evaluating LDR.

Another approach to reducing simulation time is presented in [Berrojo 2002a, Berrojo 2002c]. Through the use of rules derived from the structure of the design certain faults can be shown to be equivalent to others faults or to be dominated by other faults. For example, a fault that affects an upstream flip-flop directly feeding a downstream flip-flop through a shift register of N stages, is equivalent to a fault in the downstream flip-flop, N cycles later. Faults that affect flip-flops on primary outputs need not be evaluated, as they obviously propagate. Additional rules can be derived based on the stimulus patterns. For situations where read and write operations can be clearly identified, any faults injected between two writes with no intervening reads can be eliminated, as they will not propagate. When these techniques were applied to a circuit used in a spatial application, the static techniques allowed the total fault list to be collapsed to ≈13% and the dynamic techniques allowed a further reduction to ≈4.7%. Fault collapsing techniques are attractive to prune the number of faults; the remaining faults can then be executed in an intelligent simulation environment. In the cited work, checkpoint techniques are also described which avoid repeatedly running the portion of the simulation prior to the time when the fault is injected as well as avoiding simulating beyond the point when the system returns to its golden state.

3.1.2.2 Emulation Based Fault Injection

FPGA emulation techniques are used to accelerate fault injection campaigns [Lopez-Ongil 2007, Valderas 2007, de Andres 2008, Mohammadi 2012]. The advantage of FPGA based techniques is that the circuit can run orders of magnitude faster than in simulation. Care must be taken to ensure that the fault-injection platform does not introduce bottlenecks that limit the effective fault-injection performance. Common sources of bottlenecks are in the communication with the host computer and also in the time required to actually inject the fault.

One of the drawbacks with emulation based techniques is that it is more difficult to assess the effect of the fault. As discussed in section 3.1.2.1, an intelligent testbench can assess the type of error and classify it based on its system-level effect. On an emulation platform, it is possible to match the bit-level state of the device under test against the state of a golden design. It is thus possible to determine if there are *latent* faults in the design. It is, however, more challenging to perform run-time, high-level fault-classification as this must be performed in the FPGA hardware. As will be seen in 3.2.6, an important fraction of faults may result in an ongoing perturbation of the circuit state without causing a serious error.

Another drawback of emulation based fault-injection lies in its scalability for large designs. A full ASIC today typically has between 5-20M flip-flops and can not be mapped into a single FPGA. The routing resources available in an ASIC far exceed those of FPGAs, thus congestion and management of large-fanouts may become serious issues in an FPGA implementation, requiring extensive engineering effort to resolve. Furthermore, it is common for ASICs to have hundreds of megabits of embedded memory, well beyond what is available in FPGAs. Techniques do exist for mapping large designs into FPGAs and these involve mapping embedded RAMs to external RAMs and scaling some aspects of the design (e.g. reduced bus-widths, reduced cache-sizes, etc.). Overall, emulation techniques are best suited to designs where the full circuit under test can be comfortably mapped into a single FPGA device.

Different techniques can be used to inject faults on an FPGA-based system. One approach is to instrument the original circuit [Civera 2001]. To inject SEUs, the flip-flops where faults are to be injected in the original circuit are mapped to instrumented flip-flops. The instrumented flip-flops have additional ports that enable faults to be injected and for the results of the fault injection to be observed or checked.

To fully exploit the potential of emulation based fault-injection, the controller that injects the faults must be implemented in hardware and run natively on the card with the circuit under test. The autonomous emulation platform presented in [Lopez-Ongil 2007] has all of these characteristics and fault-injection rates of 4.2 μsec per fault are reported for test benches that require 600 clocks to execute. The largest circuits that were studied in this work had approximately 2,300 flip-flops.

A fundamentally different approach to FPGA-based fault injection is based on dynamic re-configuration of the FPGA. With this type of approach, the original circuit is not modified when it is mapped. Instead, at the point when the fault is to be injected, the clock is stopped, the FPGA is reconfigured so that the new configuration models the effect of the injected fault and then the circuit is resumed. Because the host-controller is involved in the fault injection, there is an inherent speed penalty with this approach. Results presented in [Antoni 2002] report that it takes approximately 1 msec per experiment.

There is growing concern about the effects of SETs, even in terrestrial applications. Emulating the effect of SETs is significantly more difficult than for SEUs. First, the actual gates in the target circuit implementation do not exist in the FPGA but are instead implemented using Look Up Tables (LUTs). Secondly, the delays in the FPGA implementation are completely different from those in the target implementation. These issues were addressed by researchers at Carlos III University through a quantized time approach [Entrena 2009]. The Standard Delay Format (SDF) file for the gate-level netlist is analyzed and a minimum time-step is identified. In the target FPGA, the circuit is then mapped with shift-registers replacing wires and by selecting the correct stage on the shift register, arbitrary delays can be emulated, with a resolution down to the selected time quantum. Transient pulses of arbitrary width can be injected by changing the value on the delay chain, for

the required number of time quanta. The fine-grained temporal emulation is only required to propagate the effect of the SET until it reaches one (or many) sequential elements. In [Entrena 2012], the AMUSE system is presented where both an RTL and a gate-level implementation of the circuit under test coexist in the FPGA. The RTL model runs nearly one hundred times faster than the gate-level model and is used prior to the fault-injection and after the SET has reached the memory elements. The gate-level version, is used for the short window in time when the SET is injected. Using this platform, SET fault injections can be performed on a full Leon2 processor at a rate of approximately 3000 faults per second.

These techniques require additional engineering effort. The limited size of FPGAs makes it challenging to deal with large designs and the level of analysis that can be performed when a failure occurs is limited by the "intelligence" embedded in the emulation platform.

3.1.2.3 Software Based Fault Injection

When studying processor based systems and when the actual hardware is available, it is possible to use code running on the processor to inject faults that model the effect of SEUs. This technique is referred to as Software Implemented Fault Injection (SWIFI) or Code Emulated Upset (CEU). The execution sequence of a typical software based fault injection platform is reproduced from [Valderas 2007] in figure 3.4.

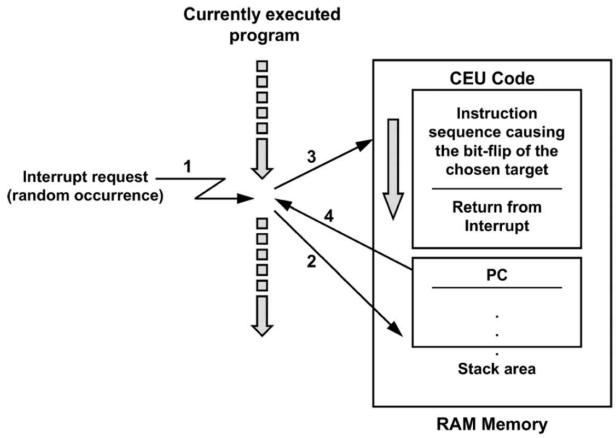


Figure 3.4: Execution Flow for Software Based Fault Injection

The normal application is run on the processor and then, at a randomly selected time, the processor is interrupted, typically using a Non-Maskable Interrupt (NMI), so that faults can be injected at any instruction in the program. The interrupt routine then selects a bit to be flipped, amongst those that are accessible to software. Typically these include bits in the register file, the cache, the program counter as well as the code, data and stack segments. The interrupt routine terminates and

execution of the application resumes. When designing a SWIFI experiment, the set of bits to inject must be carefully considered. For example, if the cache is ECC protected, it does not make sense to inject faults in it.

When performing SWIFI experiments, multiple Error Detection Mechanisms (EDMs) can be used [Arlat 2003]. Of course, at the end of the application, the final output of the program must be checked. Additionally, processors also include embedded hardware checks such as the detection of bus errors or illegal op-codes. Depending on the system, external hardware checking may be also present in the form of watchdog timers. The operating system and firmware are also able to detect certain errors through protocol checks and checksums on packets. When designing the experiment, the failures detected by the EDMs must be mapped to meaningful categories, based on the application.

An alternative approach to injecting software faults is used in [Arlat 2003] and consists in modifying the compiled program (code segment or static data segment) prior to its execution. This approach has the advantage that the program flow is not interrupted.

Overall, the key advantage of SWIFI or CEU techniques is in the ease of implementation and the fact that the application can run at real time in its target environment. In this way, the interactions with the overall system are accurately modeled. The main disadvantage is that not all state bits are directly accessible by software, thus only a subset of all of the real faults can be studied.

3.1.2.4 Comparison of Fault Injection Techniques

Given the variety of Fault Injection (FI) techniques, it is critical to understand how the results obtained with different techniques compare. In an early study [Stott 1998], SWIFI techniques were compared with bit-level fault injection on a cycle accurate processor model. The application was a network interface card where the main component was a processor running a program called MCP. In these experiments, the faults were specifically targeted at the instruction currently being executed. Exactly the same set of faults was injected in both the SWIFI and the simulation platform. The results of this study are reproduced in the table 3.2. The second and third columns show the number of faults that resulted in each category using the simulation and SWIFI techniques. The column on the right shows the percentage of times that the two techniques produced the same result.

The results are interesting in that both techniques are effective at identifying which faults produce no error (99.5% correlation). The correlation is also quite good for less severe failures such as dropped messages and corrupted data. For the more severe errors, the correlation is much weaker. Given the relatively small number of faults that were analyzed in this study, the errors bars are significant and broad conclusions can not be drawn.

Other studies [Cardarilli 2002] on small designs have shown a very high correlation between CEU techniques, flip-flop level fault injection and radiation test results. In this work, two application programs were run on an 8051 micro-controller and the

Table 3.2: Breakdown of Number of Errors By Category

Fault Injection Result	Simulation	SWIFI	Match
MCP Hang	14	51	19.6%
Hang Remote MCP	0	5	0.0%
MCP Restart	14	15	33.3%
Message Dropped	58	54	94.4%
Data Corruption	29	19	78.9%
No Error	270	241	99.5%
Total	385	385	83.5%

error rates obtained from CEU and from simulation based fault injection matched to within a few percent.

In [Arlat 2003], an in-depth comparison of errors produced by radiation, pin-level forcing and electromagnetic interference versus SWIFI techniques was performed for an application running on a 68070 processor. This work showed that bit-flips in the code-segment are able to produce error profiles similar to the hardware injection mechanisms, however, certain hardware level EDMs were not able to be triggered by SWIFI. Furthermore, when the application was protected with dual execution, SWIFI underestimated the fraction of silent failures compared to the actual radiation testing.

A recent study [Cho 2013] compares the results of fault injection at the micro-architectural and program level versus direct fault injection in the flip-flops. In the micro-architectural level simulations, either a random bit is flipped in a register file (RegU) or a random bit is flipped on a write operation to the register file (RegW). For the application level faults, either a single bit error is injected into the value of a program variable (VarU) stored in memory or a single bit error is injected into a piece of data being written to memory (VarW).

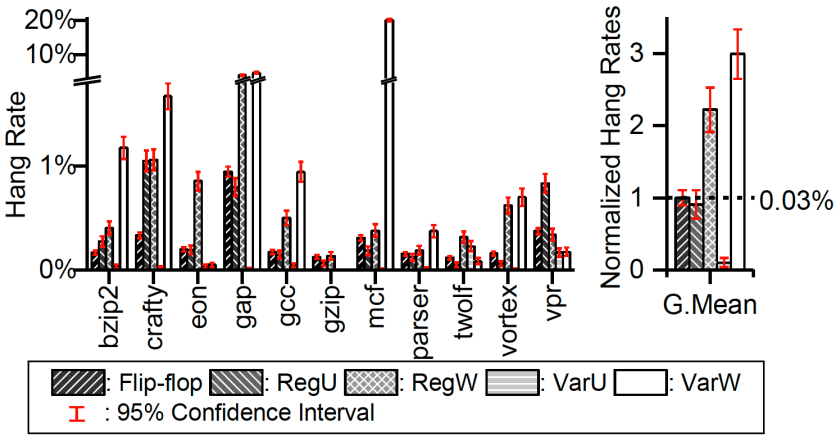


Figure 3.5: Results of Different Types of Fault Injection

The target system was a Leon3 processor and multiple complex applications were used as workloads running for billions of clock. An excerpt of the results is reproduced in figure 3.5. From these results, a difference of a factor of three is observed in the mean rate of *hangs*. The authors conclude that high-level fault injection can be highly inaccurate compared to flip-flop error injection.

Overall, care must be taken when using SWIFI to predict the actual rate of errors induced by soft error effects. First, we note that there are many different approaches to SWIFI and there is are significant variations in the error effects they produce. In some cases, SWIFI can provide a reasonable indication of the overall error rate, however, there is evidence that it can not predict the distribution of the types of errors actually produced by SEUs.

3.1.3 Analytical Techniques for Processors

An analytical technique for assessing the sensitivity of processor designs was introduced in [Mukherjee 2003] and further developed in [Mukherjee 2008]. This approach classifies the bits in a processor based on whether they are necessary for Architecturally Correct Execution (ACE). A bit that is ACE is important for the program to produce correct output. The notion of Architectural Vulnerability Factor (AVF) is defined as the ratio of ACE bits to the total number of bits in a hardware structure. To some extent, AVF is a term, introduced at Intel that has the same meaning as LDR. A structure with an AVF of '1' or a LDR factor of '1' is sensitive to all upsets.

Many bits in a processor can quickly be classified as un-ACE. For example, units used for branch-prediction are inherently un-ACE as mis-predicting a branch does not affect the program's results. Certain instructions are inherently un-ACE such as NOPs and instructions used only to pre-fetch data into the cache. In addition, the results of many instruction are never actually consumed. These results represent First-level Dynamically Dead (FDD) data and are un-ACE.

Architectural and performance models for a processor are usually used for AVF analysis and can provide results about the sensitivity of the different functional units (e.g. re-order unit, ALU, etc.). These models run very quickly, and it is possible to evaluate the AVF of real applications that run for millions of cycles.

The occupancy of the queues, such as an instruction queue, plays an important role. When a queue is nearly empty, upsets to its memory locations have a low probability of propagating. Conversely, when a queue is full, almost all of its bits are sensitive. For this reason, Little's Law[1] is sometimes used to estimate the steady-state AVF of structures containing queues.

One of the drawbacks with AVF analysis is that there is no distinction between the masking that is intrinsic in the hardware, and the portion of the masking that is introduced by the application. In [Sridharan 2009], the notion of Program Vulnerability Factor (PVF) was introduced to separately capture that portion of the

[1] Little's Law states that the long-term average depth of a queue depends on the average arrival rate and the average waiting time.

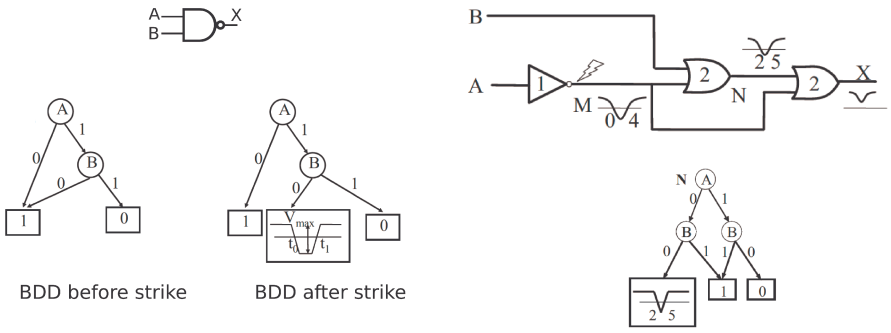
masking that is dependent on the application.

In recent work [George 2010], rigorous fault injection experiments have shown that ACE analysis sometimes overestimates the vulnerability of some structures up to 7x. ACE analysis can predict the propagation of critical bits for simple structures but is unable to predict all of the masking effects that occur in complex structures. This is especially true of faults which can not be accurately modelled with a performance simulator. Furthermore, ACE analysis only considers logical masking effects (LDR) and is unaware of TDR. Despite these drawbacks, ACE analysis has numerous benefits especially since it can be performed early in the design cycle, before detailed RTL models are available.

3.1.4 Analytical Circuit Level Techniques

Several researchers [Zhang 2006a, Miskov-Zivanov 2006, Bhaduri 2007, Miskov-Zivanov 2010] have studied the use of Binary Decision Diagram (BDD) [Bryant 1986] and Algebraic Decision Diagram (ADD) [Bahar 1993] representations of logic circuits in order to analyze the effects of faults. The intended goal of these approaches is to reduce the compute effort required to evaluate the effects of SEUs and SETs compared to fault-injection simulations (either at gate or SPICE level).

The FASER tool [Zhang 2006a] computes the effective error rate for a combinatorial network for faults induced by SETs, taking into account LDR and TDR. The tool uses BDDs to represent the logic network. In the framework of this tool, fault injection simulations using SPICE are first used to characterize the soft error sensitivity of each cell in the library, for each of its input combinations. Then, to evaluate the effect of a given fault, the BDD for the impacted gate is updated and this is propagated through the network. The modified BDD for a NAND gate with a fault affecting the B input is reproduced from [Zhang 2006a] in figure 3.6(a). The new terminal node contains information about the width and amplitude of the pulse, for the case (A=1, B=0) where a $0 \rightarrow 1$ fault can propagate from the B input.



(a) Modified BDD for a Gate (b) Modified BDD for a Network

Figure 3.6: Modified BDDs representing SET faults

At its core, the FASER algorithm iterates over each internal node of each gate

in the circuit including each combination of input patterns for the gate. Based on the condition being evaluated, the BDD for the impacted gate is updated and then the BDD for the upstream portion of the full network is rebuilt. An example of an updated network BDD is shown in figure 3.6(b). The delays on each gate of the network are considered, so cases where reconvergent paths cause a pulse to be cancelled out are correctly evaluated. The authors claim that the correct modeling of reconvergent paths is essential for the accuracy of the results, however, no actual data is provided to justify the claim. In order to process larger circuits, this tool partitions the network and treats each partition separately. The authors show strong correlation between the results of FASER analysis with fault injections performed using SPICE for the full network and claim a huge speedup compared to SPICE simulation. However, the FASER algorithm still involves an exhaustive evaluation of the propagation of each fault for each input state. Inherently, the accuracy of FASER is no better than that which would be obtained using SDF annotated gate-level event-driven simulation and it is not clear that the use of BDDs is faster than gate-level simulation. Due to the exhaustive evaluation of all faults, this approach is not scalable to large circuits.

The MARS-C tool, described in [Miskov-Zivanov 2006], uses a combination of BDDs and ADDs to simultaneously model LDR, TDR and EDR. The authors start by identifying a temporal window during which a transient must arrive at the circuit output in order to meet the setup/hold window and be sampled.

For each gate, an initial ADD is created representing the duration and amplitude of a SET occurring in that gate. Models developed by [Omana 2003] are used to define how the amplitude and duration of a transient are modified as they propagate through intermediate gates. Using the initial ADD, representing the faults, and the pulse transformation rules, a full set of ADDs are generated to model the propagation of faults in each gate to each output. A terminal value of '0' in the ADD represents all the cases where the transient is completely masked (either electrically or logically). In figure 3.7, an ADD showing the propagation of a fault occurring in gate G2 through to the output of G5 is shown. In this figure, *D235* represent the transformation of the initial pulse at G2 as it passes through G3 and G5.

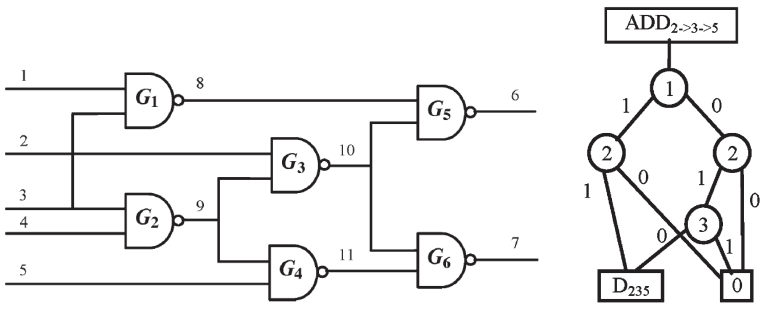


Figure 3.7: ADD Representing the Path from G2 to G3

An algorithm to accurately model the effect of reconvergent paths when building the ADDs is used and it takes into account the arrival times of the pulses on each path and the controlling values of the reconvergent gate. Once the ADDs have been built, the total error rate for the entire circuit can be calculated. The complexity of the algorithm grows with the product of the number of outputs and the number of gates in the circuit. This tool is able to evaluate circuits with approximately 500 gates in a few minutes of CPU time, which is impressive given that all of the relevant de-rating factors are considered and the reported accuracy is very close to that obtained using SPICE.

3.1.5 Formal Techniques

Formal techniques consisting of model-checking and symbolic simulation are increasingly used in the context of functional verification. A model checker is able to exhaustively explore all the states that are reachable from a set of initial states. In the context of verification, a typical application of these techniques is to ensure that a design respects a certain formal property such as being free of deadlock. Several researchers [Seshia 2007, Shazli 2008, Darbari 2008, Baarir 2009] have looked at using these techniques to assess the effect of faults on circuits as well as to identify critical flip-flops.

In [Seshia 2007], a SpaceWire communication circuit with 130 latches is analyzed. The Verilog description of the circuit was manually translated into a format readable by the SMV tool [Cadence 1998]. The authors then analyzed the english specification for the SpaceWire protocol and translated the requirements into 39 formal assertions that define the correct operation. The initial model was first validated to conform to the specification in the absence of faults. A formal model for a SEU was created. The model expresses that a given bit can change state at an arbitrary cycle. The model checker was then rerun repeatedly, each time with the formal model for a SEU in each of the latches. The authors found that upsets in only 28 of the 130 latches would cause a violation of the specification. A single run of the model-checker required less than 4 minutes. The approach is interesting because after protecting the 28 critical latches, the design is known to operate correctly under *all* situations. Unfortunately, the engineering effort required to create the formal specifications is quite high and requires experts familiar with SMV.

The approach presented in [Seshia 2007] makes it possible to determine whether a given state bit is critical for correct operation. It does not, however, provide a means to rank the critical state bits. In [Baarir 2009], the authors introduce the notion of repairing states. When a fault is introduced in the formal model, after some number of incorrect states, it may return to the correct state. The ratio of the number of repairing states to the total number of reachable states provides an indication of the criticality of a state bit. The design considered in this work is a single state-machine with six states and small number of inputs and outputs.

Formal techniques for the analysis of SEU are certainly of academic interest, however, it does not appear that they can be applied to industrial scale designs,

such as those considered in the second half of this chapter.

3.1.6 Probabilistic Techniques

Probabilistic techniques for analyzing fault propagation through combinatorial networks have been well studied and two early tools to perform this type of analysis were PROTEST [Wunderlich 1985] and STAFAN [Jain 1985]. The approach is based on first computing the probability, $P(i)$, that each signal in the network has the value '1'. Next, the probability that a fault on any signal will propagate to the outputs is evaluated. Typically, a probability of 0.5 is assigned to the primary inputs of the circuit. Then working forward through the network, the signal probabilities, P(i) are propagated from the inputs to the outputs of the gates using, the relations shown in figure 3.8.

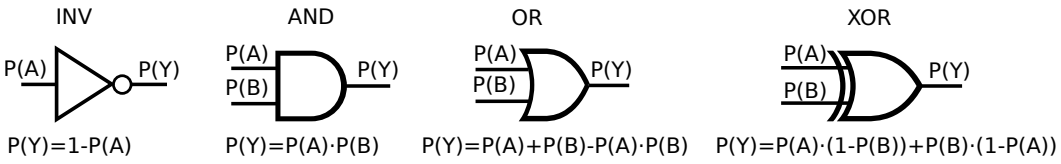


Figure 3.8: Forward Propagation of Signal Probabilities

Next, the detection probability of faults is calculated. These are computed separately for faults causing a signal to transition to '1', denoted $DP1(i)$, and those causing a signal to transition to '0', denoted $DP0(i)$. This is done working backwards from the primary outputs. The detection probabilities, $DP0(Y)$ and $DP1(Y)$ of a primary output, Y, are simple $P(Y)$ and $1 - P(Y)$, respectively. Then, using the relations shown in figure 3.9, the detection probabilities can be propagated backward through the network.

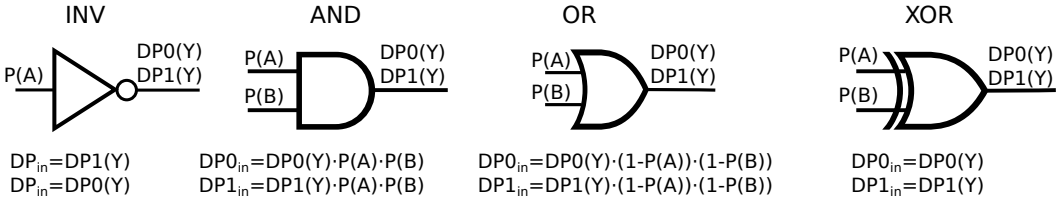


Figure 3.9: Backward Propagation of Detection Probabilities

The classic problem with probabilistic approaches is how to handle reconvergent paths as shown in figure 3.10. In this example, at E, conflicting detection probabilities are propagated backward from D and from C. To handle this case, a common assumption is to take the maximum of the detection probabilities from the upstream nodes, although this overestimates the detection probability.

Beyond simply using probabilistic analysis techniques to compute the LDR of the gates in a network, recent work [Hayes 2007, Polian 2008] has studied how to rank

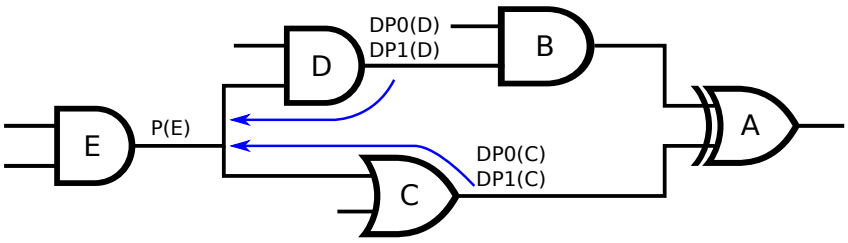


Figure 3.10: Reconvergent Paths

gates for selective hardening. In the first paper, exact solutions for computing which gates to harden are presented for small circuits. In the followup, a faster approximate technique is presented which is able to rank gates for selectively hardening and handle large circuits with over 2 million gates.

3.1.7 Comparison of Analysis Techniques

In the first half of this chapter, many techniques for analyzing the effects of soft errors have been reviewed. In order to compare the techniques a set of qualitative criteria have been identified.

1. *SEU* indicates whether the technique can evaluate the effect of SEUs.
2. *SET* indicates whether the technique can evaluate the effect of SETs.
3. *Abstraction* indicates the abstraction level of the model being analyzed.
4. *Scale* provides an indication of the technique's ability to process large designs.
5. *Speed* indicates how quickly the technique can complete an analysis.
6. *Domain* indicates the type of designs that can be analyzed with the technique.
7. *FDR* indicates whether the functional effect of faults can be taken into account.
8. *TDR* indicates whether the technique can assess the masking effects of temporal delays.

Eight different techniques were compared based on these criteria and the results are summarized in table 3.3. The row labelled 'Sim. Basic' refers to the simple fault simulation approach that is illustrated in figure 3.3(a) and the row labelled 'Sim. Adv.' refers to the simulation approach shown in figure 3.3(b). The difference between these techniques lies in how the results are analyzed. Through the use of an intelligent test-bench, it is possible to assess the functional effect of faults. Although simulation based fault injection runs slowly, there is no hard limit on the size of designs that can be simulated.

To conclude, simulation based fault injection provides the most general means to evaluate the effect of faults. Emulation is attractive when the scale limitations

are acceptable. Analytical techniques have an important role in specific cases; for example AVF is valuable when evaluating processors at the architectural level.

Table 3.3: Comparison of Soft Error Effects Analysis Techniques

Technique		SEU	SET	Abstraction	Scale	Speed	Domain	FDR	TDR
Fault Inject	Sim. Basic	✓	✓	RTL/Gate	High	Slow[c]	General		✓[b]
	Sim. Adv.	✓	✓	RTL/Gate	High	Slow[c]	General	✓	✓[b]
	Emulation	✓	✓[a]	RTL/Gate	Medium	Fast	General		✓[a]
	Software	✓		Hardware	High	Faster	Processor	✓	
Analysis	AVF	✓		Arch.	High	Fast	Processor	✓[e]	
	BDD/ADD	✓	✓	Gate	Low	Medium	General		✓
	Probabilistic	✓	✓	RTL/Gate	High	Fast	General		
	Formal	✓		RTL/Gate	Low	Medium	General		

[a] Most approaches to fault emulation do not consider temporal delays. Recent work has shown techniques to model SETs [Entrena 2012] and to consider the effect of TDR [Ebrahimi 2014] in emulation.

[b] TDR can only be modeled in simulation with a gate-level netlist and a SDF file.

[c] Through acceleration techniques, the speed of fault injection simulations can be improved.

[d] Most research on probabilistic fault propagation is at the gate level, but recent work [Chen 2011] has studied RTL-level propagation.

[e] In some work using formal techniques, [Seshia 2007], the requirements are expressed formally. In this way, only faults that cause the functional specification to be violated are flagged as errors and FDR is thus taken into account.

3.2 Networking Case Study

In this section, the results of a detailed analysis of the effects SEUs in flip-flops for three large design blocks totalling over a half million flip-flops taken from a 40nm commercial NP ASIC is presented. The objective of the analysis is to determine the expected rate of specific classes of system failures when a large population of routers is deployed in a network. The RAMs in these designs were fully protected by ECC and memories with large interleaving were selected to avoid MBUs so flip-flop SEUs were the dominant SER failure mechanism. Table 3.4 shows the flip-flop counts and major function of the three blocks that were studied.

Table 3.4: Summary of Design Blocks

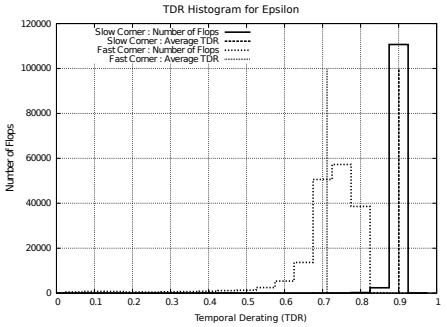
Name	Function	Number Flops
epsilon	Output data-path	102 559
gamma	On-chip packet storage	341 615
omega	Packets re-assembly from DRAM	184 552

3.2.1 Temporal De-Rating

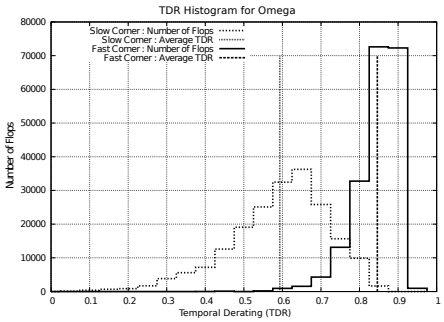
The concept of TDR was introduced in section 1.3.2. Given the scale of the design blocks, performing fault-injections with a gate-level netlist is not practical. In RTL level simulation, there are no delays thus the effect of TDR is not modelled. For this reason, the TDR analysis is done separately from the fault-injection analysis. The TDR values for the flip-flops in all three blocks were calculated using Static Timing Analysis (STA) with Synopsys Primetime to extract the post-route timing. For each flip-flop, the delay to all end-points was obtained and the ratio of the path with the greatest slack to the clock period was used to calculate the TDR. This process was repeated for all flip-flops both at the fastest and the slowest timing corners and histograms of the results for *epsilon*, *omega* and *gamma* are shown in figure 3.11. The TDR value is shown on the abscissa and the number of paths with that TDR value is shown vertically. Table 3.5 shows the average temporal de-rating factors for the three blocks. At the slow process corner, the TDR is numerically lower because the fraction of time that the flip-flops are vulnerable is smaller. The three blocks being studied are all located in the same clock domain. It is interesting to note, that even at the slow timing corner, the effect of TDR is less than a factor of two.

3.2.2 Classifying Failure Effects

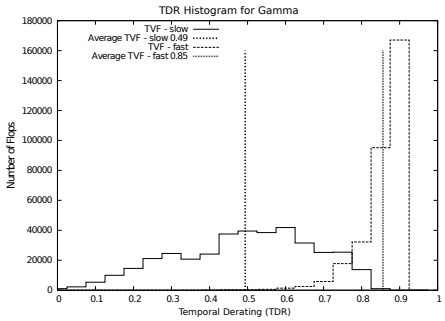
A classification of soft errors effects for micro-processors is proposed in [Nguyen 2005] based on the following categories: 1) masked errors , 2) correctable errors, 3) DUE and 4) SDC. This classification was adapted to the networking application.



(a) Epsilon TDR



(b) Omega TDR



(c) Gamma TDR

Figure 3.11: TDR Distribution in Epsilon, Omega and Gamma

It is important to make a distinction between two cases. A SEU may be logically masked, meaning that after 1..N clock cycles, the errored state is over-written and the design returns to the logical state which it would have had if the error had not occurred (e.g. *golden state*). An error can also be functionally masked, meaning that the circuit state remains permanently altered, however, the resulting output sequence still meets all the system level requirements. This could be because the upset occurs in a piece of circuitry that is not active and has no effect or because the output sequence is slightly modified, but is still correct. In some work, these cases are viewed as latent faults [Berrojo 2002b], since it is possible that some set

Table 3.5: Summary of Average TDR Values

Block	**Average TDR (Slow)**	**Average TDR (Fast)**
epsilon	0.71	0.90
gamma	0.49	0.85
omega	0.59	0.85

of stimuli might eventually sensitize the modified state and produce a failure. From manual analysis of the simulations that produced *functionally masked* outcomes in the circuits under study, it was observed that these cases typically corresponded to upsets in debug-type circuitry not used in mission mode. It is interesting to note that a fault analysis technique that only looks at the structure of a circuit, without knowledge of the intended function would not be able to identify the functionally masked scenarios.

Some SEUs are corrected due to ECC logic. This is the case of flip-flops on a memory data-path after the ECC generation and prior to the ECC check and correct logic. The corrected case is distinguished from the logically masked case, because in the blocks under study, correctable ECC errors result in an interrupt being flagged. This interrupt causes the testbench to "fail", as it does not expect to see these interrupts during the simulation. In the field, however, upsets in these flip-flops would be indistinguishable from SEUs in the actually RAM.

Some SEUs are explicitly detected through mechanisms such as parity. These errors are not correctable and require a reset to bring the device back into normal operation. Such failures are classified as being *detected explicitly* since it is clear that they are the result of a soft error effect. Sometimes, however, a SEU may provoke a detectable error through a side-effect which is classified as being *detected indirectly*. For example, if a FIFO pointer is corrupted, it may trigger the overflow detection logic for the FIFO. In the system, such an interrupt would result in a chip reset and traffic would resume after an outage. In these cases, it is not possible to attribute the cause of the interrupt to a soft error effect and in the field such an event might be misdiagnosed as defective hardware or as the occurrence of a rare hardware or software bug.

The remainder of the failures are deemed *silent errors*, however, the special case where the effect is a one time corruption of a packet or a statistics counter is explicitly called out. Although this is not a desirable outcome, it may go unnoticed in a core network application. The other *silent error* cases are more serious, where internal hardware state such as pointers, linked-lists, etc. are corrupted and where the chip is likely to start operating in an unpredictable fashion. In certain cases, the packet traffic ceases to flow, resulting in a testbench timeout, and these are classified as *silent lockups*. Table 3.6 summarizes the fault classifications.

3.2.3 Testbench Environment

The functional verification of blocks within an ASIC is normally done using a test-bench environment with random stimulus and scoreboard-based result checking. Since the result checking is done at a high level of abstraction (e.g. packets, transactions), the testbench is tolerant of minor changes in the output sequence, provided the functional requirements are still met. If the testbench determines that the design is behaving incorrectly, an error message is reported in the simulation log file and usually the simulation is terminated.

In order to associate test-bench failures to the resulting system effect, each of the

Table 3.6: Classification of SEU Effects for Networking Application

Category	Sub-Category	System Outage	Description
Masked	Logically	None	Upset state is overwritten after one or more clocks.
	Functionally	None	Upset state remains but has no functional effect.
Corrected		None	Error is corrected (e.g. ECC).
Detected	Explicitly	Medium	Explicit mechanism detected the error (e.g. parity).
	Indirectly	High	Error detected due to a side-effect. (e.g. FIFO overflow).
Silent	Minor	Low	Operation is silently affected, but impact is contained (e.g. single corrupted packet).
	Major	Unknown	Internal state is affected causing unpredictable effects (e.g. linked list corruption).
	Lockup	Unknown	Packet traffic ceases to flow.

error messages reported by the testbench environment must be manually mapped to one of the categories in table 3.6. For example, if an injected SEU triggers a testbench error due to an unexpected FIFO overflow interrupt, then it would classified as an *indirectly detected* SEU effect. This mapping is performed by building a set of rules using *regular expressions* that matched the error messages in the simulation log files. In this way, one rule can be used to match a series of similar error messages. Using these rules, the result of a fault injection campaign can be analyzed to determine the likelihood of occurrence of different SEU effects.

3.2.4 Simulation Speedup and CPU Efficiency

In a typical functional verification environment, the RTL description of the block is simulated using a testcase that goes through a series of timeline steps in order to reset the device, perform configuration, apply stimulus, stop the stimulus and then audit the state of the design (e.g. check all FIFOs are empty, check the value of all packet counters, etc.). It must be ensured that the SEU injections occur during the window of time when the design is fully active and processing data so that the results are representative of a system operating in the field. It is wasteful to re-simulate the entire reset and configuration sequence for each fault simulation and the first optimization consists of storing the simulator state just prior to the

window in time where fault injections can be performed. A single checkpoint is stored and when injecting a fault at a later time, this checkpoint is loaded and the simulator advances up to the desired injection time. It is possible to store multiple temporal checkpoints [Berrojo 2002b] to avoid simulating the design from the time of the checkpoint to the time of the fault injection, however, there is a trade-off between disk-space and CPU. In section 3.2.4, it is shown that this is not where the majority of the CPU time is spent.

The simulations for this study were performed using Synopsys VCS and a SystemVerilog testbench. The checkpoints were created using the save/restore facility of VCS controlled by a UCLI TcL script. The flipping of the flip-flop states was accomplished with a VPI routine. The flip-flops to upset were selected using a random uniform selection (with replacement) from the set of flip-flops present in the RTL. The timing of injection was selected uniformly randomly over fixed window of 100 clock cycles during which the design is fully active. For each block, three test cases were selected which taken together spanned the full mission-mode functionality of the design and all the modes of operation. For each testcase, three golden runs were performed using different random seeds in order to include a rich set of stimulus with different packet sizes, different packet rates and variability of other configurable parameters. For each fault simulation, the selection amongst the nine scenarios was also uniformly random.

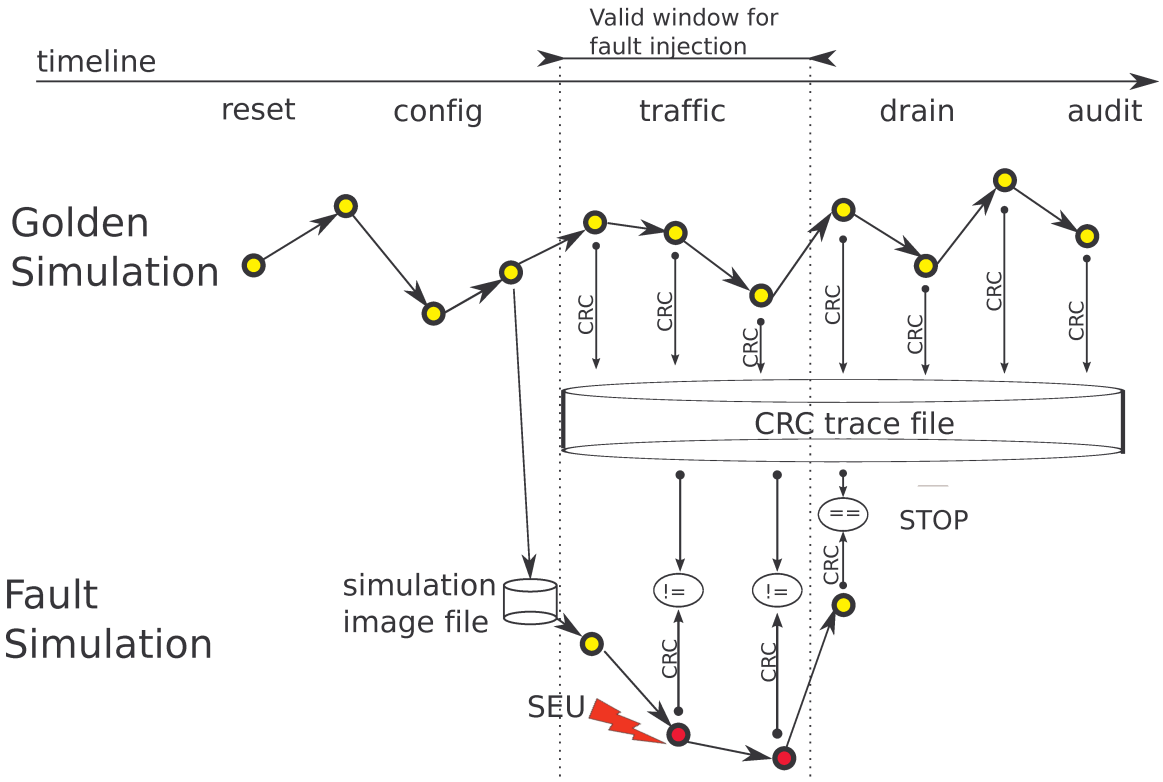


Figure 3.12: Testbench with Checkpoint and CRC Matching

It is well known that a large fraction of SEUs are logically masked because they are either not captured in a downstream flip-flop during the cycle of the upset or because after some number of cycles, the errored state is overwritten [Berrojo 2002b]. After this occurs, the internal design state has returned to the *golden state*. Continuing to run the simulation beyond this point is wasteful. In order to determine that the design has returned to the golden state, we need a means to compare the golden state with the state during a fault simulation. Storing the full design state over a large number of clocks requires significant disk space and accessing the data introduces an I/O bottleneck. Instead, we propose a new technique where a Cyclical Redundancy Check (CRC) is calculated over the entire design state (both flip-flops and memories) and stored in a text file file during a series of initial golden simulation runs. During the fault simulation, periodically after the SEU injection, the same CRC is computed over the design state and compared with the value stored in the text file during the golden simulation. If the values match, the design has returned to the golden state and the simulation can be terminated, with the knowledge that the outcome would have been a PASS, had it run to completion as illustrated in figure 3.12. If multiple state bits in the fault simulation differ with the golden simulation, there is a small risk that the CRC-32 could alias and produce a false match. The probability of an alias never exceeds 1 in 2^{32} and thus represents a much smaller source of error than that which is intrinsic from the statistical sampling of the faults.

There is a compute overhead to calculating the CRC over the design state. Since we know a large fraction of the faults disappear quickly, we chose to compute the CRC every clock cycle for the first 100 clocks and then only every 100^{th} clock cycle for the remainder of the simulation. By initially checking the state every clock, we are able to accurately determine how quickly faults are overwritten in the window after the fault injection. Beyond this initial window of time after the fault, there is little to be gained from frequently comparing the design state. In the worst case, a simulation will continue to run 99 clocks after its state has re-converged, before the next comparison. In [Berrojo 2002b], a similar approach was used and the period for performing the checking was increased exponentially after the fault injection.

The combination of both techniques (checkpointing and early termination on state matching) provides an average speed-up for fault injection simulation ranging between 8x to 12x. Faults simulations that produce a testbench failure as well as those that re-converge to the golden state terminate early and therefore require less CPU time. In fact, during a fault injection campaign, a large fraction of the CPU time is devoted to simulating faults that don't re-converge to the golden state but which produce no error as shown in table 3.7. Although the passing cases with no golden match represent between 7%-15% of the simulation runs, they account for between 40%-55% of the CPU time.

3.2.5 Logical De-Rating Results

Logical de-rating occurs when the corrupted value resulting from an SEU is overwritten. This can occur in the clock cycle of the upset, which we refer to as LDR_1,

Table 3.7: CPU Usage During Simulation Campaigns

Category		Epsilon		Gamma		Omega	
		Runs	CPU %	Runs	CPU %	Runs	CPU %
Fail		2102	26.5	1536	32.4	3312	33.8
Pass	Matched	7148	32.6	7439	25.0	4858	10.8
	No Match	750	40.9	1025	42.6	1427	55.4

or it can occur after N clock cycles, LDR_N. Our simulation environment gives us visibility into the full design state so we know LDR_N for different values of N. The abscissa of the graph in figure 3.13 is the number of clock cycles after the upset event. On the vertical axis, the cumulative percentage of simulation runs that have converged with the golden state is shown for each of the three blocks. It can be observed that a significant number of upsets never get sampled (N=1) and it is interesting to note that this percentage varies significantly between *omega* (23%) and *epsilon* (45%). A second observation is that some upsets remain present for thousands of clock cycles before being overwritten. The cases where the upset persists for an extended period of time require more time to simulate.

3.2.6 Summary of Simulation Results

Table 3.8: Observed Effect of SEUs (as percentages)

Category	Sub-Category	Epsilon	Gamma	Omega
Masked	Logically	71.5 ± 0.8	74.4 ± 0.9	60.7 ± 3.2
	Functionally	7.5 ± 0.5	10.3 ± 0.6	13.9 ± 2.2
Corrected		0.9 ± 0.2	0.6 ± 0.2	0.8 ± 0.6
Detected	Explicitly	0.1 ± 0.1	6.1 ± 0.5	6.7 ± 1.6
	Indirectly	13.1 ± 0.6	1.0 ± 0.2	0.3 ± 0.4
Silent	Minor	5.6 ± 0.4	3.0 ± 0.3	11.6 ± 2.1
	Major	1.3 ± 0.2	4.7 ± 0.4	4.1 ± 1.3
	Lockup	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 0.9

The results of the fault injection simulations are summarized in table 3.8. The confidence intervals were computed using equation 3.3. The distribution of SEU effects between the blocks is fairly similar, however, it is clear that *epsilon* is, overall, a less sensitive design. *Gamma* and *omega* manipulate complex data-structures such

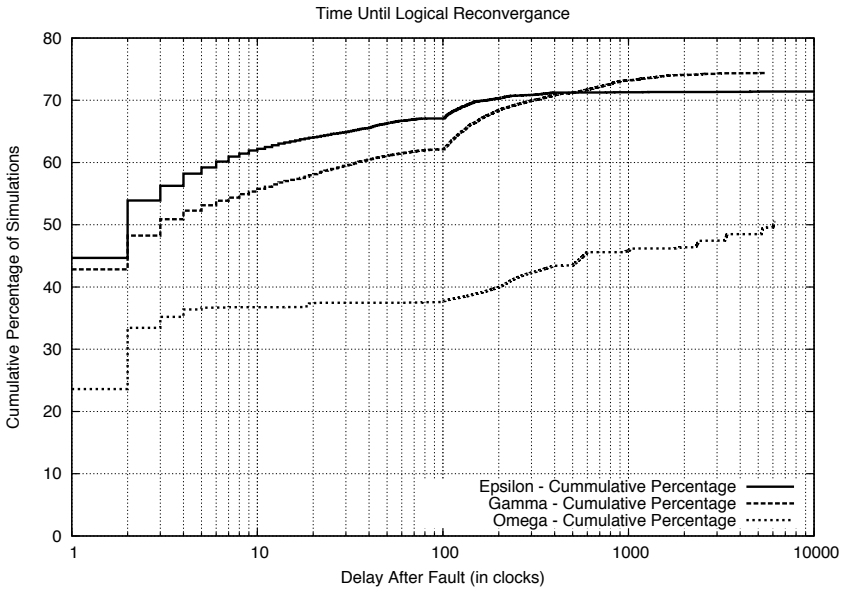


Figure 3.13: Logical Re-Convergence Versus Time After Fault Injection

as linked lists and free-pools and it is thus more likely that SEUs will have a serious effect. These blocks also have some parity-protected flop-based structures, thus a larger fraction of SEUs are explicitly detected.

3.2.7 Combining the Results

Returning to the Internet router system level specification [Silburt 2009], the most critical outage category are those failures that produce an *unknown* effect which have been classified as *silent major* and *silent lockup* in table 3.6. From the results of the fault injection simulation campaign, we know the fraction of SEUs that produce such failures. As a safe bound, we know this fraction can be further reduced by the TDR at the fast timing corner and these results are summarized in table 3.9.

Table 3.9: Percentage of SEUs Producing Serious Outages

Name	**Percentage SEUs Producing a Serious Impact**
epsilon	1.2 %
gamma	6.5 %
omega	5.2 %

These results, combined with the technology soft error rate, can be used to predict the expected types of failures in the field. The data in table 3.9, also suggests that only a small fraction of flip-flops produce serious outages and thus selective hardening of this small set of flip-flops could produce a significant improvement in the rate of serious failures. Extension of this work to select critical flip-flops is discussed in chapter 4.

3.3 Statistical Analysis of Fault Injection Results

Statistical Fault Injection (SFI) provides a means to estimate the probability that faults cause errors. The variance of this estimate depends on the number of experiments that are performed and when presenting results of SFI campaigns, it is essential to report the error bars.

When we perform a series of fault-injection simulation runs in order to estimate the probability, p, of a given outcome, we are performing a series of *Bernoulli* trials. A *Bernouilli* trial is a random experiment that can have one of two outcomes. If the i^{th} simulation run produces the given outcome, we define the random variable $X_i = 1$ otherwise $X_i = 0$. It is intuitively clear that, $\hat{p}$, the best estimator of p, after n trials, is given by equation 3.1. After each simulation we can compute $\hat{p}_n$ and plot the value to observe the convergence trend.

$$\hat{p}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{3.1}$$

In our experiments, the full sample space consists of all the flip-flops in the block ($\approx 10^5$) sampled over a temporal window of 100 clocks considered for nine workloads and is thus large ($N \approx 10^8$) compared to the number of simulation runs ($n = 10^5$) and therefore we assume an infinite population size[2].

The variance of a Bernoulli random variable is $p \cdot (p-1)$. There are different ways to estimate a confidence interval for a given value of α which is the level of confidence. $\alpha = 0.05$ for a 95% confidence interval. A very loose confidence interval which does not depend on the central limit theorem is given by Hoeffding's inequality [Wasserman 2005, p. 65] as shown in equation 3.2.

$$\hat{p}_n \pm \sqrt{\frac{1}{2n} log(\frac{2}{\alpha})} \tag{3.2}$$

If we assume that the distribution of $\hat{p}_n$ is normal, which for large values of n is true due to the central limit theorem, then a tighter confidence interval [Wasserman 2005, p. 130] is given by equation 3.3. The interval depends on $\hat{p}_n$, the current estimate of p after n simulations. $z_{\alpha/2}$ is the abscissa value where the area under standard normal distribution, up to that point, is equal to $1 - \alpha/2$ and can easily be found

[2] We can apply a finite population correction factor of $\sqrt{\frac{N-n}{N-1}}$ to the confidence intervals, however, for a large sample space this is ≈ 1.

in standard tables [Trivedi 1982]. $z_{alpha/2}$ is 1.96, 1.65 and 1.28 for 95%, 90% and 80% confidence intervals, respectively.

$$\hat{p}_n \pm z_{\alpha/2}\sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}} \tag{3.3}$$

For small values of n, this bound is tighter if the student-T distribution [Johnson 2000, p. 213-214], is used in equation 3.3 instead of $z_{\alpha/2}$ as shown in equation 3.4. For large values of n, the student-T distribution converges to the normal distribution.

$$\hat{p}_n \pm t_{\alpha,n}\sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}} \tag{3.4}$$

The disadvantage with these two confidence bounds is that the interval depends on the estimated value of p. In [Leveugle 2009], it is observed that the largest interval occurs when $p = 0.5$, and thus a conservative confidence interval is proposed by always substituting $p = 0.5$ in equation 3.3, instead of the current estimated value of p.

In order to evaluate the different confidence intervals, we plotted $\hat{p}_n$ versus the number of simulations, with each of the confidence intervals. The analysis was only performed for $n \geq 10$ since with fewer than 10 samples it is difficult to draw accurate conclusions.

Since we only ran a finite number of simulations, we take the final estimate of p after 10,000 to be the "correct" value. In figure 3.14, we plot the estimated value of $\hat{p}_i$ through the course of 10,000 simulation runs, where p is the probability that an SEU will be logically masked in *epsilon*. We see that most of the time the confidence interval does indeed cover the final estimated value of 71.45%, although for n between 30 and 50, the upper bound falls below the final estimate.

This analysis was performed for each of the SEU effects identified in table 3.6 for each of the three design blocks. For each confidence interval, we computed the percentage of the time the final estimate of p is captured within the given confidence interval, through the course of the simulation campaign. These results are summarized in table 3.10.

Table 3.10: Results of Different 95% Confidence Intervals for Epsilon

	Hoefding Inequality	**Normal with** $p = 0.5$	**Normal with** $p = \hat{p}_n$	**Student T with** $p = \hat{p}_n$
epsilon	100%	99.6%	98.4%	97.6 %
gamma	100%	100%	99.1%	96.8 %
omega	100%	100%	98.9%	96.8 %

In figure 3.15, we plot all four intervals for the estimated probability that an SEU is *Indirectly Detected* in *epsilon*. From the results, we see that all four intervals do indeed provide a 95% confidence interval for the estimated value of p. The

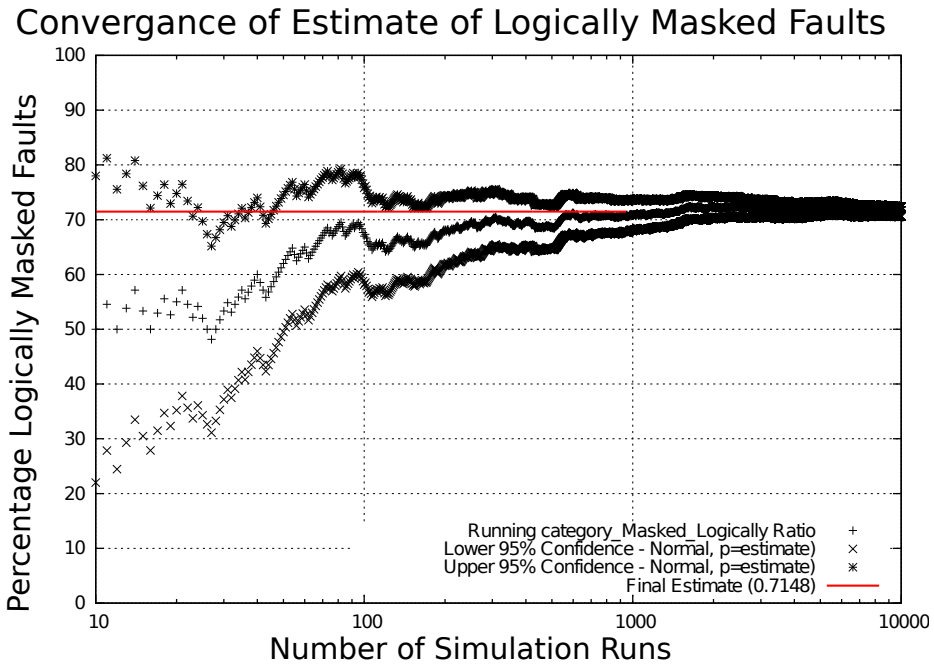


Figure 3.14: Normal Confidence Interval with $p = \hat{p}_n$ for Epsilon

Hoefding inequality provides a very weak bound, but in all cases it covers the final estimate. Using a normal distribution with $p = 0.5$ provides a tighter bound that nearly always covers the estimated value. The student-T distribution provides the tightest interval and it is still correct over 95% of the time.

3.4 Conclusions

There is a very broad body of work that addresses the problem of analyzing whether faults manifest themselves as errors, and if so, as what type of errors. In this chapter, we presented a brief overview of the existing work on fault effect analysis as well as a detailed case study of the effect of faults on large design blocks taken from a NP.

Part of the simulation based analysis included the development of a new, fast technique to quickly determine when a faulty simulation has re-converged with the golden trace, in order to avoid unnecessary simulation cycles. An analysis of the CPU time usage during the simulations showed that the majority of the CPU time is devoted to simulating cases where a latent fault persists until the end of the simulation. The case study also included an analysis and comparison of different error bounds on the estimate error rates. Most importantly, this case study presented a detailed SER analysis for design blocks much larger than those typically studied in the published literature.

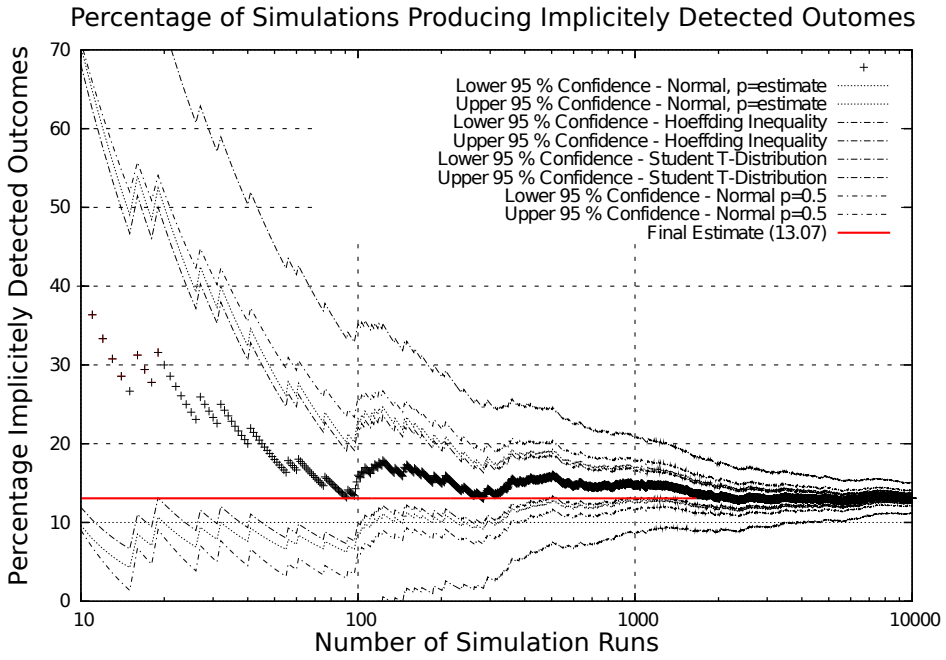


Figure 3.15: Comparison of Four Confidence Intervals

Chapter 4

Techniques for Clustering Critical Flip-Flops

Contents

4.1 Introduction

As seen in the case study presented in chapter 3, flip-flops are a major contributor to the overall soft error budget of large integrated circuits. It is well known that selectively hardening a small percentage of the flip-flops can significantly reduce the effective error rate. In commercial processors used in server applications, selective hardening is common practice and this has been well documented for the Itanium family of processors [Naffziger 2005, Stackhouse 2008]. The challenge is to identify the best set of flip-flops to harden.

Current process technologies make it possible to integrate tens of millions of flip-flops into a single SoC and within a few years, chips will have over a hundred million flip-flops, as shown by the trends illustrated in section 1.4. Thus, the effects

of flip-flop soft errors can not be ignored and decisions related to SER mitigation can significantly influence overall architecture [Bhuva 2011].

The silicon costs are minimized by protecting only those flip-flops which are critical. The definition of critical is application specific. For example, in a video application, the perturbation of a single pixel is less critical than the loss a full frame. In order to identify a minimal set of flip-flops for protection, it is important to consider the system effect of errors.

The identification of the optimal set of flip-flops to protect typically requires compute-intensive fault-injection campaigns. In this chaper, new techniques are presented which group similar flip-flops into clusters to reduce the number simulations that are required. With these techniques, the number of fault injections can be significantly lower than the total number of flip-flops. In one industrial design with over 100,000 flip-flops, by simulating only 2,100 fault injections, the technique identified a set of 4.1% of the flip-flops, which when protected, reduced the critical failure rate by a factor of 7x.

In commercial applications, the goal is not to drive the failure rate to zero, but rather to achieve a sufficient increase in reliability to meet a system requirement while minimizing the overall cost. There are three aspects to the cost : silicon cost (increased area and power), engineering cost (additional human effort) and compute cost (fault injection simulations). Often a 2x-5x reduction in the rate of critical failures is sufficient to bring a system within specification.

4.1.1 Proposed Clustering Techniques

In chapter 3, an overview of existing techniques for identifying flip-flops for selective mitigation was presented. One reason that current approaches do not scale is that the analysis is performed at the granularity of individual flip-flops. At this low level, key information about function is lost. As shown in figure 4.1, large circuits are built from groups of flip-flops constituting buses and modules that form a hierarchy.

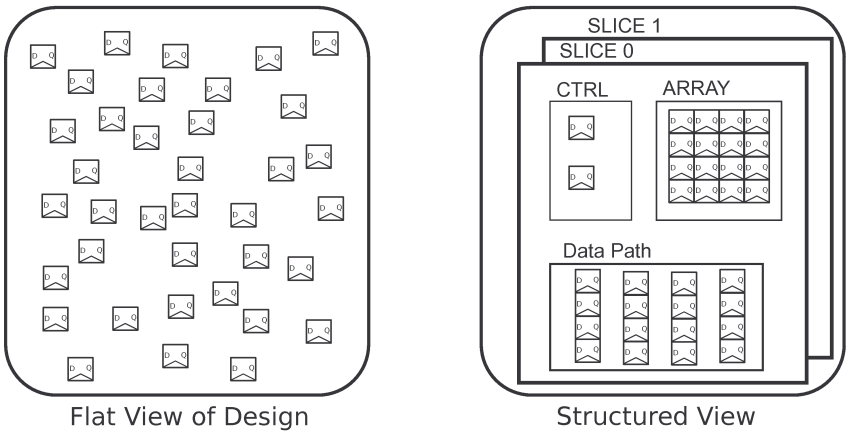


Figure 4.1: Flat and Structured View of a Design

The proposed methodology is based on coarse-grained fault injection as shown in figure 4.2. Starting with the full set of flip-flops (i), they are grouped into clusters using static information (ii). Statistical fault injection is performed for each of the clusters to rank them based on their sensitivity (iii) and then a set of the most sensitive clusters are selected for mitigation (iv). Clustering is done prior to any simulations being run and three techniques for clustering have been analyzed :

- Based on buses
- Based on design hierarchy
- Hybrid approach using buses, hierarchy and signal naming information

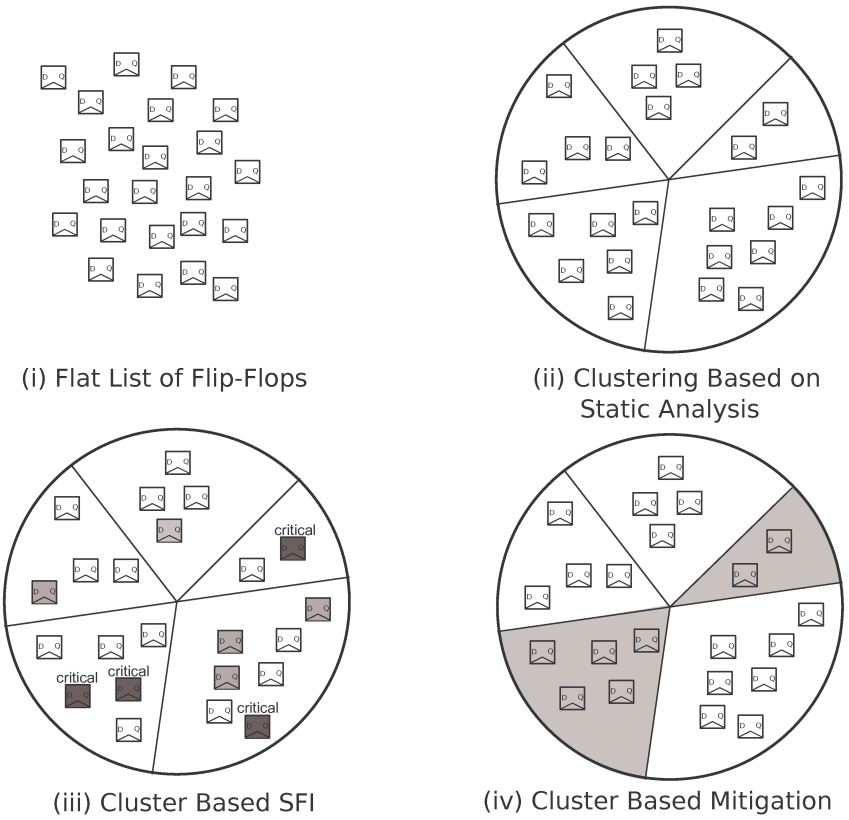


Figure 4.2: Cluster Based Selective Mitigation

The techniques were applied to an open source 10GE MAC design [Opencores 2013] and to two of the large blocks from an industrial NP that were studied in chapter 3. The designs are summarized in table 4.1.

4.1.2 Flat SFI Results

In order to provide an objective measure of the sensitivity of each of the original designs, a *flat* SFI campaign was performed. All simulations were performed at

Table 4.1: Summary of Design Blocks

Name	Functional Description	Number Flip-Flops
10GE MAC	Simple 10G Ethernet MAC.	1,138
Epsilon	Data-path with header processing.	102,559
Omega	Complex packet assembly from DRAM.	184,552

the RTL level employing the environment used for functional verification making it possible to evaluate the system-level impact of errors. For the industrial designs, a selection of three testcases (e.g. workloads) was used and for the 10GE MAC, the supplied Verilog testbench was used. The focus of this analysis is on SEUs in flip-flops, so the fault injection mechanism consisted of inverting the value stored in a flip-flop (Verilog *reg*) using a simulator VPI routine. The list of flip-flops was obtained from a synthesized gate-level net-list and projected back onto the corresponding RTL signals. The time of the fault was randomized uniformly over the active region of the test-case. For the *flat* simulation campaign, the selection of the flip-flops was uniform.

Different networking applications have different reliability requirements. In the public Internet, important data is protected with strong checksums, thus an infrequent payload corruption can be tolerated. If the effect of the SEU causes the circuit to stop forwarding data or to start continuously corrupting data, then it is deemed *critical*. The latter failures are highly problematic and the goal of mitigation is to reduce their rate of occurence.

When routers and switches are used in private data-centres or in High Performance Computing (HPC) systems, the reliability constraints are different. Here, SDC is not acceptable, as the packets are carrying data between compute nodes and there are not necessarily application level checks on the data.

The designs studied in this section are from Internet networking applications and the results of the simulations are classified into three categories as shown in table 4.2. In these applications, if the effect of the SEU is a one-time corruption of a packet, it is deemed *minor*.

The results of the *flat* fault-injection campaign are presented in table 4.3. 90% confidence intervals are computed using equation 4.1 where n is the number of simulations and $\hat{p}_n$ is the estimated probability [Wasserman 2005, p. 130]. It was also confirmed visually that the error rates had converged. The probability of *critical* outcomes is low, but the absolute rate of critical failures remains unacceptably high. The focus of the mitigation effort is to minimize the *critical* failure rate.

$$\hat{p}_n \pm z_{\alpha/2}\sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}} \tag{4.1}$$

Table 4.2: Simulation Result Classification

Pass	Test-bench passes. The upset was masked.
Minor Impact	Test-bench fails. Error messages show the effect is a one-time payload corruption or there is a detectable interrupt.
Critical Impact	Test-bench fails and the error messages indicate that packets have stopped flowing or are being continuously corrupted.

Table 4.3: Sensitivity of Original Design Blocks

Name	**Number Faults**	**Percentage Masked**	**Percentage Minor**	**Percentage Critical**
10GE MAC	10 000	76.1±0.8%	21.9±0.8%	**2.2±0.4%**
Epsilon	10 000	79.1±0.8%	19.6±0.7%	**1.4±0.2%**
Omega	5 000	69.1±1.3%	19.1±1.1%	**11.7±0.9%**

4.2 Static Clustering Techniques

Real-world designs are structured and contain buses, module hierarchy and often have repeated instances of the same module. This information is extracted through analysis of the RTL code and exploited to group the flip-flops into clusters likely to have similar functions and thus a similar sensitivity to errors. Three clustering techniques are outlined below followed by the results of targetted SFI campaigns.

4.2.1 Bus Based Clustering

In the HDL description of large circuits, most registers are declared as a bus or vector (e.g. reg[31:0] PC) and have a similar function. For example, all the bits within the program counter of a processor would be equally sensitive to upsets. In the designs under study, over 90% of the flip-flops are part of a bus and the average bus size is between 14 and 20 bits. Table 4.4 shows the number of buses extracted from each design. The size of the clusters is not uniform and it is observed that a few large clusters dominate, therefore, the number of clusters that must be considered to cover 80% of the flip-flops is presented as an indication of the number of large clusters.

The main problem with this approach is the discrete flip-flops which are not part of a bus. In all the designs, these represent less than 10% of the flip-flops, however, these are often control signals and thus critical.

Table 4.4: Bus Based Clustering

Design	Number Clusters	Average Size	Clusters for 80%	Flops Not in a Bus
10GE MAC	59	18.6	22	37 (3.3%)
Epsilon	5,314	20.2	1,580	2,883 (2.6%)
Omega	12,737	14.1	5,427	18,409 (9.3%)

4.2.2 Hierarchical Clustering

In large designs, certain modules implement functions related to the control path while others contain large data-paths. It is considered good design practice to separate control logic from data-path logic to customize synthesis constraints. It is important to note that certain modules contain non-mission mode logic for debug and thus have a low sensitivity to SEU effects. For these reasons, it is possible to cluster the flip-flops based on the module instance they belong to, as different instances may have different functions. The results of applying hierarchical clustering are summarized in table 4.5. Not surprisingly, there are fewer, larger clusters.

Table 4.5: Hierarchy Based Clustering

Design	Number Clusters	Average Size	Clusters for 80%
10GE MAC	10	114	3
Epsilon	580	190	111
Omega	1285	154	401

4.2.3 Hybrid Clustering

The third clustering scheme is an extension of the two previous ones. A distance function is defined to evaluate the "proximity" of two flip-flops. The function is defined so that flip-flops in the same bus and flip-flops which have the same signal name, but which are in different instance hierarchies (e.g. *a.b.c.block_enable* and *d.e.f.block_enable*) have a small distance. Pairs of flip-flops whose names differ by only one or or two characters (e.g. *a.b.c.r_ptr* and *a.b.c.w_ptr*) also have a small distance. This heuristic makes it possible to group signals differing only in their suffix (e.g. *wr_en_a* and *wr_en_b*) that likely have similar functions. As the differences in hierarchy and signal name increase, so does the inter-flop distance function.

Using the distance function, clustering is performed using an agglomerative clus-

tering algorithm [Everitt 1993]. Initially, there is one flip-flop per cluster; then those clusters whose distance is minimal are iteratively merged until a specified number of clusters is achieved. Currently, the target number of clusters was selected manually to ensure that similar signals were combined but to prevent unrelated signals being clustered. For example, read and write pointers from multiple FIFOs were combined into a single cluster. The results of applying the hybrid clustering algorithm are shown in table 4.6 and it is seen that large clusters are acheived for the industrial designs.

Table 4.6: Hybrid Clustering

Design	Number Clusters	Average Size	Clusters for 80%
10GE MAC	93	12.2	23
Epsilon	209	527	37
Omega	409	485	82

4.3 Simulation Results

A clustered fault injection campaign was performed using each technique on the three designs. To both ensure that large clusters are classified accurately and to avoid running excessive simulations for small clusters, the following policy was established: for a cluster containing N flip-flops, $N/5$ fault-injections were performed, with a minimum of 3 and a maximum of 15 runs. This policy allocates more simulation runs to larger clusters in order to minimize the risk of mis-classifying a large cluster. With 15 fault injections, the risk of mis-classification is very low, thus by establishing a ceiling, unnecessary runs are avoided.

4.3.1 Bus Based Clustering

The bus-based fault-injection campaign on the 10GE MAC required 266 simulations and the clusters were sorted based on the number of critical failures that were observed. Eight clusters produced at least one critical failure and these clusters represented 167 flip-flops. The results are shown in the histogram in figure 4.3. The clusters are binned based on the fraction of simulations that produced a critical failure, which is a measure of the sensitivity and shown on the horizontal axis. The black bars on the histogram show the number of clusters having a given sensitivity and the grey bars on the right show the number of flip-flops contained in these clusters. The climbing line indicates the cumulative number of flip-flops contained in the clusters starting with the most sensitive clusters on the left side. Note the number of flip-flops is shown on a log scale.

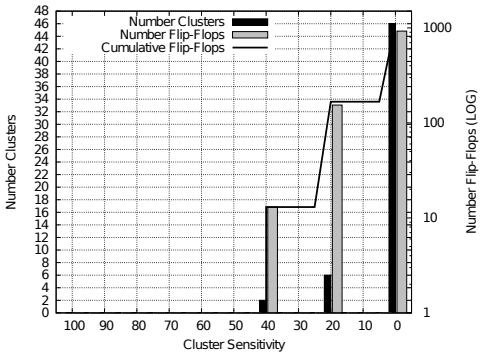


Figure 4.3: Bus Based Clustering for 10GE MAC

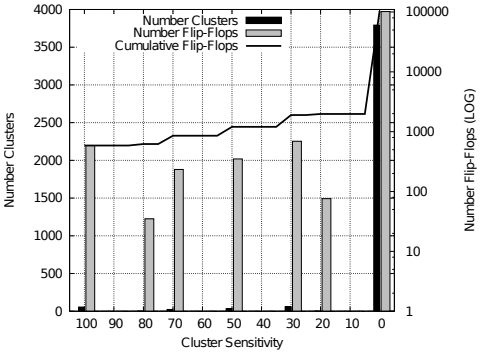


Figure 4.4: Bus Based Clustering for Epsilon

A similar fault-injection campaign was performed on Epsilon, with bus-based clustering. 12,324 simulations were required and the results are shown in figure 4.4. There are many buses in Epsilon that are sensitive, producing a critical simulation outcome on nearly every fault.

4.3.2 Hierarchical Clustering

The results of hierarchical clustering on the 10GE MAC (see figure 4.5), show that the design is too small to use module hierarchy to cluster critical flip-flops. However, with hierarchical clustering on Epsilon (see figure 4.6), after 4,284 simulations it was found that only 59 out of 580 module instances, representing 14,471 flip-flops (14.1%), resulted in any critical failures.

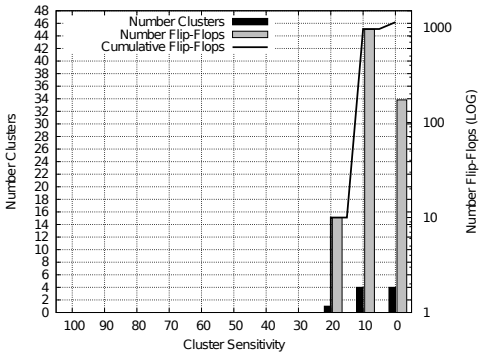


Figure 4.5: Hierarchical Clustering for 10GE MAC

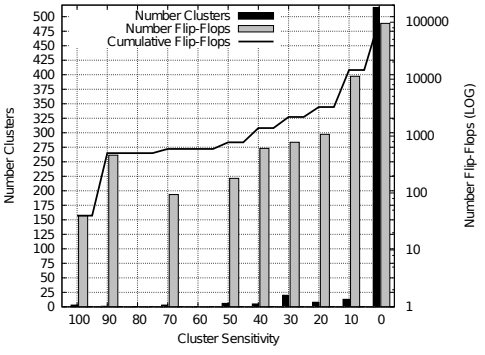


Figure 4.6: Hierarchical Clustering for Epsilon

4.3.3 Hybrid Clustering

When hybrid clustering was applied to the 10GE MAC, only 11 of the 93 clusters had critical failures and these represented 188 flip-flops (16%) as shown in figure 4.7.

When hybrid clustering was applied to Epsilon, after only 2,100 simulations, it was found that just 24 of the 209 clusters, representing 4,248 flip-flops (4.1%), resulted in a critical failure, as shown in figure 4.8. The results for Omega are presented in figure 4.9. This design unit processes data-structures based on linked-lists, so it is expected that many sensitive clusters are observed.

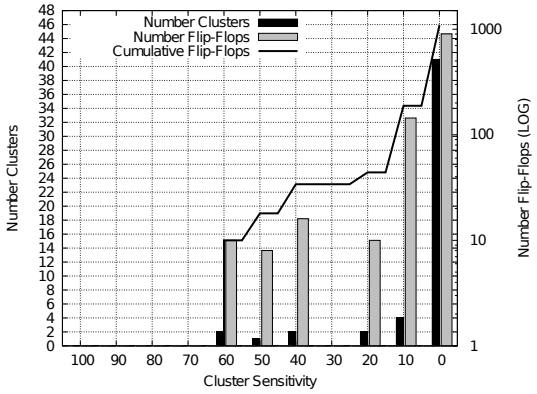


Figure 4.7: Hybrid Clustering for 10GE MAC

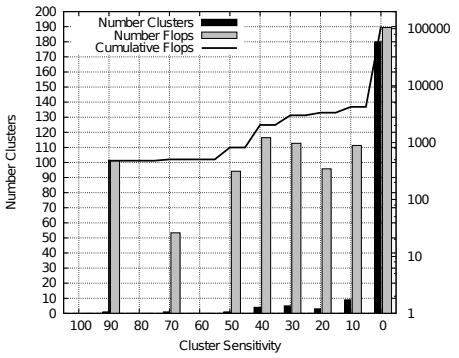


Figure 4.8: Hybrid Clustering for Epsilon

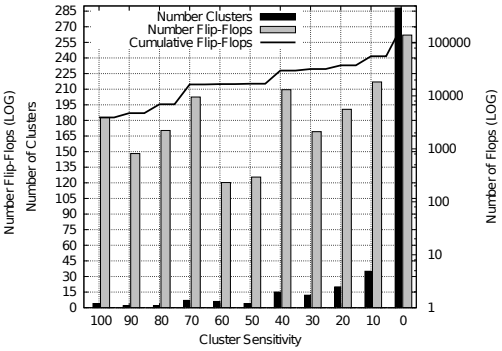


Figure 4.9: Hybrid Clustering for Omega

4.4 Selective Mitigation

The results from the clustered fault-injection runs can now be used to develop mitigation policies. A policy consists of selecting an approach to protect a set of flip-flops. The protection could be achieved by substitution with hardened cells, TMR or other other types of hardened sequentials discussed in chapter 2. For simplicity, it is assumed that after mitigation, the SEU sensitivity is zero, but this assumption is easily relaxed.

The sensitivity of the hardened design can now be evaluated without any further simulations by re-analyzing the results from the original *flat* campaign. Where one of

the mitigated flip-flops was randomly selected in the original campaign, the analysis is updated as if the outcome had been a *pass*. This approach is valid because the flat campaign has run without any knowledge of the clustering and thus the sampling is uniform across the design.

In order to clearly validate the new approach, an *independent* flat analysis was performed. In fact, the flat analysis is not strictly necessary and it is possible to simply perform a single, clustered fault injection campaign. The results of this campaign can be used to both assess the initial design sensitivity and to assess the sensitivity of the mitigated design. This simplification comes at the expense of an independent verification of the results.

4.4.1 Full Mitigation

The simplest policy is to just protect all the flip-flops in all the clusters which showed any critical failures. This provides an upper bound on the level of improvement that can be obtained. Table 4.7 shows the reduced critical sensitivity (the new percentage and the factor reduction compared to the original results in table 4.3) and the percentage of protected flip-flops to obtain the improvement.

Table 4.7: Reduction in Sensitivity with Full Mitigation

	Bus Based		**Hierarchy**		**Hybrid**	
Design	Critical	Flops	Critical	Flops	Critical	Flops
10 GE	1.12±0.17%		0.39±0.10%		0.70±0.14%	
MAC	2.0x	15%	5.5x	84%	3.1x	16%
Epsilon	0.44±0.16%		0.30±0.15%		0.18±0.10%	
	3.5x	1.9%	4.3x	14%	7x	4.1%
Omega	3.15±0.49%		1.47±0.28%		0.74±0.24%	
	3.7x	13.8%	7.8x	36%	14.9x	39%

4.4.2 Partial Mitigation

In practice, the goal is to achieve a reliability target with a minimum cost so it may be unnecessary to protect all clusters. The effect on the sensitivity of the designs was evaluated with various percentages of flip-flops being protected. The flip-flops to protect were selected starting with the most sensitive clusters first. The results for Epsilon and Omega are shown in figures 4.10 and 4.11. The error bars show the 90% confidence interval for the mitigated design. It is clearly possible to trade-off soft error sensitivity with the number of protected flip-flops and the steeper slope on the left shows that the most critical clusters were identified. It is important to note that these multiple design points were obtained from the analysis of a single clustered fault-injection campaign.

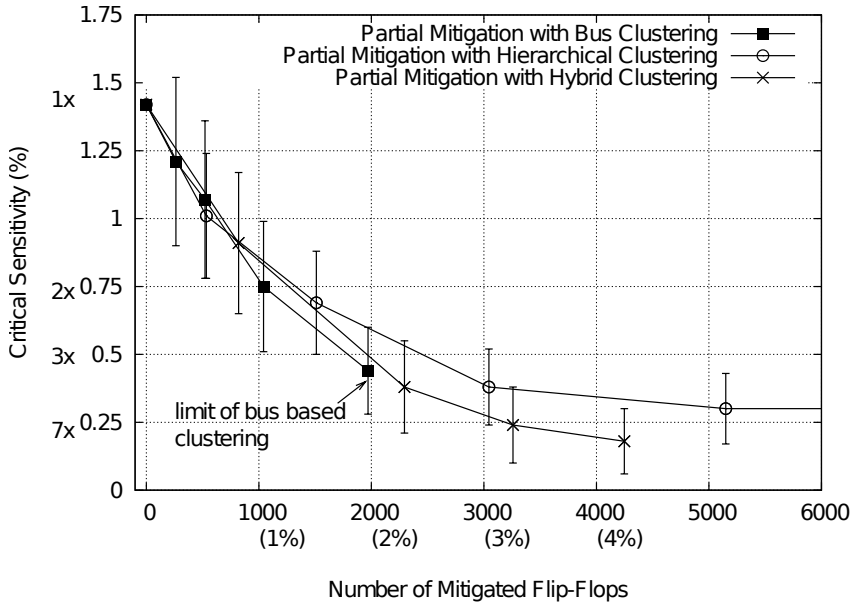


Figure 4.10: Partial Mitigation in Epsilon

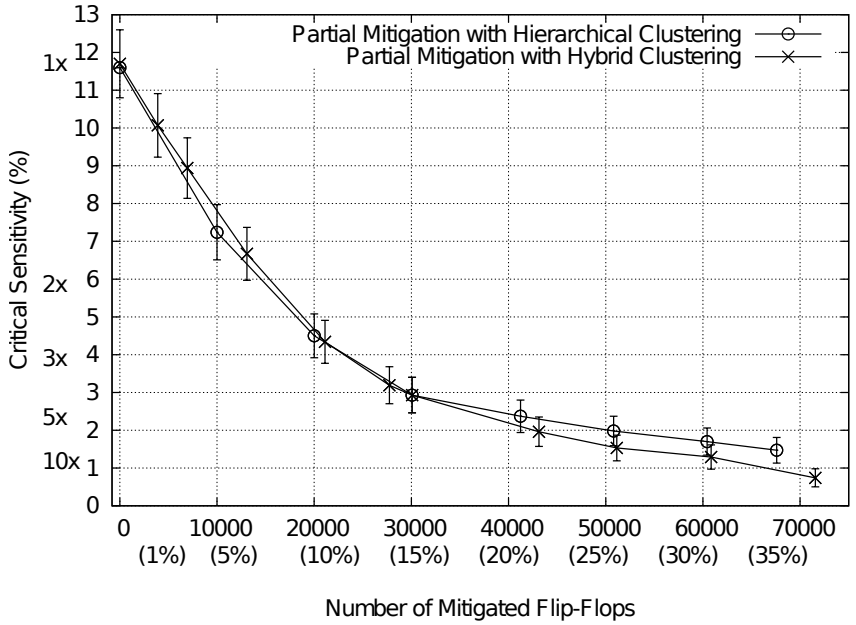


Figure 4.11: Partial Mitigation in Omega

Bus-based clustering provides policies with the lowest cost due to the smaller cluster sizes, however, there is a limit to the improvement that can be achieved because the discrete signals are excluded from the analysis. Due to the larger cluster sizes, hierarchical clustering provides more costly mitigation policies than the hybrid clustering.

4.5 Discussion

When performing fault injections, the fault space is three dimensional with the axes being : time, space and work-load (simulation trace), as shown in figure 4.12. Certain flip-flops are critical every clock-cycle (e.g. reset signals, interrupts, etc.) while others are only critical when specific transactions are being processed.

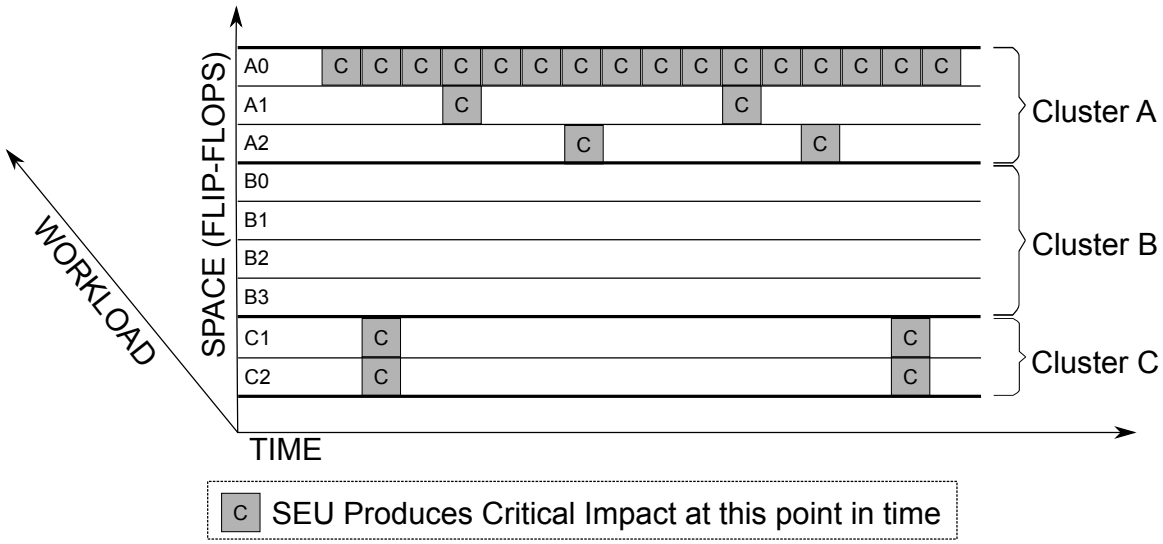


Figure 4.12: Three Dimensional Fault Space

If flip-flops are considered individually, multiple fault-injections are required to assess if there are times when they are critical. Identifying flip-flops that are sensitive at least 20% of the time, requires about five simulations per flip-flop, which is a prohibitive number. It is thus difficult to compare the current results with an optimal solution through exhaustive simulation.

With this technique, only a small number of fault-injections per *cluster* are needed to determine that some flips-flops in the cluster are critical. With cluster sizes of several hundred flip-flops, this represents a significant reduction. However, with larger clusters, there is a risk that the elements are heterogeneous and that some non-critical flip-flops are protected unnecessarily. This is seen by the higher cost associated with the policies from hierarchical clustering. Hybrid clustering was better able to group flip-flops with similar sensitivity.

The bus-based clusters are smaller but exclude flip-flops that are not bussed (10% of the flip-flops in the examples). With Epsilon, a maximum of 3x reduction in critical failures was possible with bus-based clustering compared to a 7x reduction with hybrid clustering. In figure 4.10, it is seen that with the finer granularity of the bus-based clustering, a slightly better reliability improvement is obtained, for a given number of protected flip-flops, but this comes with the cost of more simulations.

The proposed techniques are based on heuristics and the quality of the results is subject to implementation style, however, interesting results have been obtained

on both small and large designs. Without simplifying assumptions, the problem of identifying critical flip-flops in a large design remains overly compute intensive. The circuits selected here are from networking applications, however, other types of designs such as processors have similar characteristics : presence of buses, module hierarchy and reasonable signal naming conventions, thus it is expected comparable results would be obtained.

4.6 Conclusions

In this chapter, it has been shown how clustering techniques based on design structure combined with limited fault injection simulations can be used to identify sensitive nodes. No assumptions are made about the design other than the availability of a functional test-bench used to assess the impact of SEU faults. The techniques were applied to sequential circuits with hundreds of thousands of flip-flops. Using the results from a single, clustered, fault-injection campaign, a range of mitigation policies can be developed.

Future work in this area will focus on identifying improved clustering approaches that exploit the connectivity and data-flow within the design as well as applying the techniques to a broader class of circuits. Another direction of future work relates to better optimizing the number of simulations.

Chapter 5

Hierarchical Single Event Transient Analysis

Contents

5.1 Introduction

In section 1.4.1, it was shown that due to both design trends and technology scaling, SETs represent an increasing fraction of the overall SER of large SoCs. Accurately estimating this component is important but remains a challenging problem for full designs with tens or hundreds of millions of gates.

The majority of the existing work on SET analysis starts from a gate-level circuit representation, however, in an industrial design cycle, by the time a gate-level net-list is available, it is too late to make significant design changes. The hierarchical SET analysis methodology described in this chapter can be applied at the RTL level. This methodology, was initially published in [Evans 2013] using SER results obtained for the NanGate 45nm library. An extended version of the methodology was then developed under contract to a major networking company and was delivered in the form of a spreadsheet that can be used to estimate combinatorial SER. The commercial implementation used SER data obtained from simulations performed on a commercial 28nm cell library.

In this methodology, the SET sensitivity of the cell library and the masking characteristics of standard combinatorial design blocks are pre-characterized and stored

in compact models. Then, the SET sensitivity of a complex circuit is calculated by decomposing it into blocks that have been pre-characterized. These block-level models are combined taking into account the data flow, to estimate the overall SET sensitivity. Experimental results for an ALU implemented in the NanGate 45nm library as well as partial results from the analysis of a commercial 28nm cell library are presented.

The starting point for this approach is to simulate the SET sensitivity of the combinatorial cells in the library, or at least for a representative subset. These simulations are done using a SER simulation tool and, for each cell, the result is a histogram showing the FIT rate for pulses of different widths. This step is important as the sensitivity of different cell types (INV, AND2, XOR3) can vary significantly. Due to the fact that the final analysis is done at the RTL level, this large data set is then distilled into a compact summary of the SET sensitivity for different classes of cells.

Separately, the circuit and masking characteristics of standard combinatorial blocks of the type inferred by an RTL synthesis engine (e.g. muxes, adders, etc.) are evaluated.

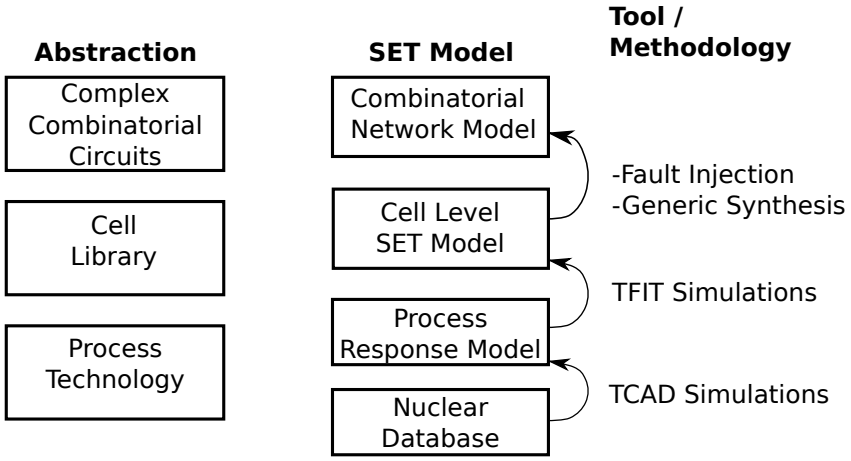


Figure 5.1: Modeling of SETs at Different Levels of Abstraction

Using the proposed methodology, the block-level models are combined with the library models to estimate the SET sensitivity of a complex combinatorial network. The proposed models are part of a chain of abstractions, as shown in figure 5.1. It is possible to output and store these models in a standard format such as RIIF [Evans 2012] which is described in chapter 6. With a standard format, it becomes possible for the intermediate models to be shared between Electronic Design Automation (EDA) tools thus automating the analysis flow.

5.1.1 Review of SET Masking

There are three reasons why SETs may not propagate and be sampled at a flip-flop : LDR, TDR and EDR. These are outlined in section 1.3.

LDR measures the likelihood that the pulse will propagate through the logic network from a boolean perspective and is technology independent. For example, an upset at the input of an adder, will propagate to the output as the sum *always* depends on the inputs. In contrast, an upset at the input to a wide multiplexer, will only propagate if the affected input is selected, thus the *error propagation probability* is much lower.

In a ripple-adder implementation, upsets on any of the gates inside the adder will always propagate. This probability of faults on the internal gates of a circuit is referred to as the *internal LDR*. If we consider a Carry Lookahead Adder (CLA), however, upsets in the carry prediction logic may not propagate, if there is no carry. It is not necessary to have a gate-level implementation in order to assess the first order effects of LDR.

In [Limbrick 2011], it is shown that the LDR, referred to as Error Propagation Probability (EPP), of a given circuit can vary based on the synthesis constraints that are applied. This work studied the effect of constraints on LDR but did *not* consider the variation in the intrinsic SER of the circuit. For example, the authors show a variation in LDR of between 0.14 and 0.20 (30%) based on varying timing constraints. However, the same constraint variations cause the cell count, and thus most likely the raw SET sensitivity, to vary by 500%. Therefore, the first order effect of synthesis constraints on SETs is the effect they have on the total gate count. The main reason for the variation, both in LDR and gate count, is certainly due to the synthesis engine selecting different circuit architectures based on the target frequency (e.g. CLA versus ripple adder). In the proposed methodology, this dependence on synthesis constraints is taken into account as different implementations of the same high-level logic function (e.g. adder) are considered to have different intrinsic SER rates and different LDR values.

To propagate, a transient must arrive at the input to one or more sequential elements when it is being sampled. TDR quantifies this probability and it was shown in section 1.3.2 that it can be effectively estimated as:

$$TDR_{SET} = \frac{\int_{w=minPW}^{w=maxPW} w \, \mathrm{d}w}{T_{clk}} \tag{5.1}$$

EDR relates to the analog propagation of the pulse and is more difficult to analyze. One aspect of EDR is accounted for by considering how the induced analog pulse is mapped to a digital pulse. The shape of a radiation induced pulse is shown in figure 5.2 and based on the logic threshold voltage, it can be modeled as a digital pulse of width PW. Pulses whose amplitude never reaches Vth can be ignored. The second aspect of EDR relates to the fact that when the pulse travels through the network, it may be attenuated or it may be stretched [Cavrois 2008] and this is not currently incorporated in the proposed methodology.

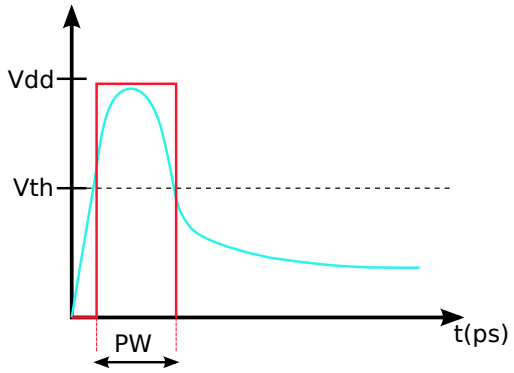


Figure 5.2: Conversion of a SET Pulse into a Digital Pulse

5.1.2 State of the Art

Many works have studied the effect of SETs, however, none has proposed an estimation methodology that starts from the RTL and that relies on pre-characterized models for high-level design blocks.

Some approaches are based on probabilistic techniques to estimate whether transients propagate due to LDR. In [Polian 2008] such a probablistic technique that consider re-convergence is presented and applied to very large circuits but neither TDR, EDR nor the intrinsic rate of SETs occuring is considered.

Other approaches are based on using on a gate-level event based simulator to evaluate the transient propagation. Algorithms to significantly accelerate gate-level transient fault simulations [Alexandrescu 2002] have been presented and applied to moderate size circuits.

A very different approach is implemented in the MARS-C platform described in [Miskov-Zivanov 2006]. The authors propose a unified circuit representation that considers LDR, TDR, EDR using a combination of BDDs and ADDs to represent the sensitive paths to the outputs. The circuits that are evaluated include ISCAS benchmarks and the results are compared with SPICE analysis. The proposed data-structures can not scale to very large circuits and this work does not consider the variation in the intrinsic SET sensitivity of different gates.

The SEAT-DA and SEAT-LA platforms described in [Rajaramant 2006] have similarities to the work in this chapter. The SEAT-DA platform is used to calculate the neturon SET sensitivity of individual gates. The SEAT-LA tool uses SPICE to pre-characterize the pulse deformation and logic masking of indidivudal gates (LDR,EDR). The two tools are combined and used to perform SET analysis on ISCAS benchmarks ($\approx$ 1200 gates) and the results are compared to SPICE simulation. The SEAT-LA platform requires 1.4 min of CPU time to analyze the SER of a 4-bit ripple adder. This framework is further extended in [Ramakrishnan 2008], where the Hierarchical Soft Error Estimation Tool (HSEET) tool is presented. Instead of characterizing the logic masking and pulse transformation for individual gates, the idea is extended to small blocks (e.g. muxes). However, with HSEET the basic

analysis still involves evaluating the propagation of pulses through a netlist and the results are presented for small circuits.

Other authors [Zhang 2006a, Rao 2006, Wang 2011, Ramanarayanan 2009] have presented accelerated SET analysis techniques but they all work at the gate level.

At the RTL level, soft errors are modeled as bit flips and their effects can be analyzed using fault injection [Gracia 2008, Wang 2004] or analytical techniques [Li 2005, Mukherjee 2005]. Some techniques [Devadas 1996, Lin 2012] have been proposed for error propagation analysis at the RTL level but they do not estimate the intrinsic rate of occurrence of SETs nor do they account for the TDR and EDR.

The remainder of the chapter is organized as follows: in section 5.2 the SET sensitivity of individual cells is analyzed. In section 5.3, the LDR of small circuit blocks is analyzed then in section 5.4 we present a complex circuit and perform a canonical analysis of its SET sensitivity. In section 5.5 we compute the SET sensitivity using the hierarchical approach and compare the results to the canonical analysis. In section 5.6 we summarize and consider future extensions to this technique.

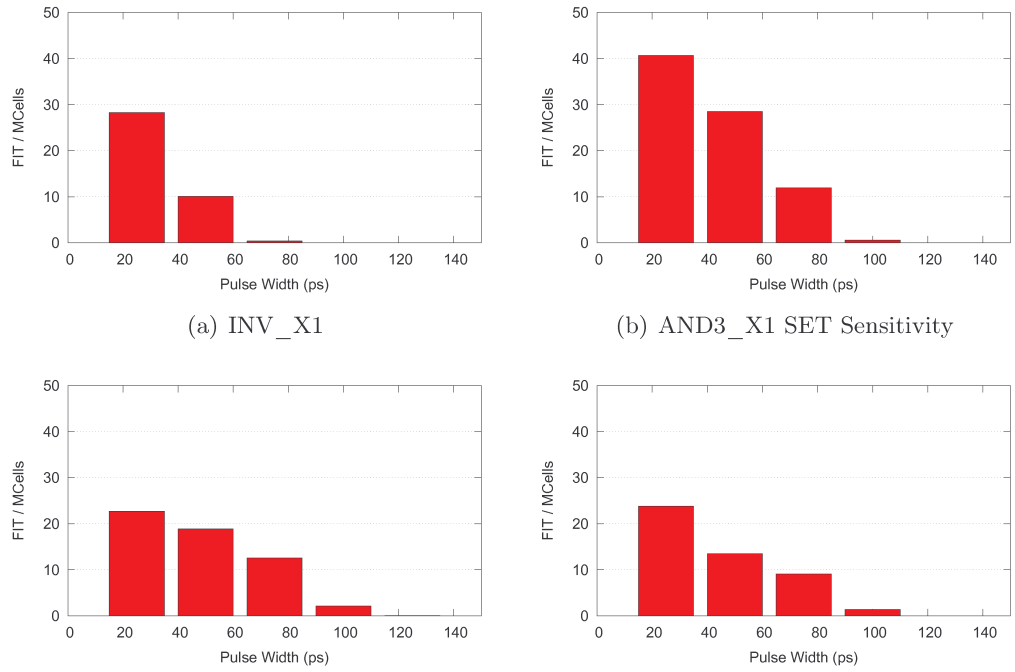
5.2 Cell Level SET Characterization

5.2.1 Technology, Cell and Library Modeling

The first step in the methodology is to characterize the SET sensitivity of the process technology and cell library. In this work, the cell-level SER analysis is performed using TFIT [Alexandrescu 2011], however, the overall methodology is not tied to this tool. With TFIT, the process technology is characterized using 3D TCAD simulations to yield a technology response model. Based on the radiative environment (atmospheric neutrons for terrestrial applications), a nuclear database is consulted to analyze the energy distribution of the particles and the nuclear interactions that occur with the materials to obtain a distribution of the different charge depositions that can occur. Separately, using the net-list and the layout for the cell, specifically-tailored transient current sources are added to the circuit netlist and simulated in Spice and the response of the cell is observed. This process is repeated for all the transistors in the circuit and all the possible circuit states. Using knowledge of the frequency of occurence of the underlying nuclear events and the circuit response obtained by Spice, a FIT rate for SETs of specific widths is computed.

Figure 5.3 shows the absolute FIT rates for pulses of different widths for four different cells taken from the 45nm NANGate Open cell library, including both the neutron and alpha contribution. In figure 5.4, the sensitivity of two different gates from a commercial 28nm library are shown. For confidentiality reasons, this data is presented with arbitrary units. In both cases, it is clear that the SET sensitivity can vary significantly between different gate types. This is further illustrated in figure 5.5 which compares the sensitivity of five different gates. Overall, in terrestrial applications, the upsets are dominated by short pulses (<=50ps).

In order to create a more compact model of the SET sensitivity of the library, the



(a) INV_X1

(b) AND3_X1 SET Sensitivity

(c) AND3_X4 SET Sensitivity

(d) AOI221_X4 SET Sensitivity

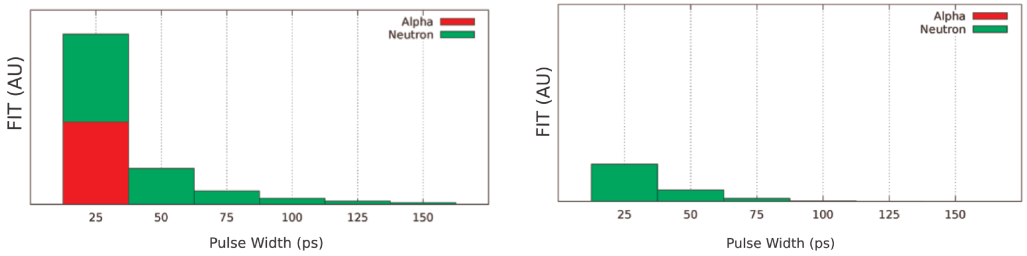
Figure 5.3: SET Sensitivity of Four 45nm Combinatorial Gates

gates were grouped into five broad categories based on function. From the RTL level, it is difficult to determine exactly which gate-types will be inferred by the synthesis engine. However, certain high-level circuits, generally map to certain categories of cells. For example, parity and ECC circuits typically require XOR gates. Logic in state-machines, typically maps to AND/OR type gates. Some libraries contain complex gates such as 2:1 multiplexers and full-adders. Since these are library specific, they form a separate group. The SER sensitivity of the library cells in each category was averaged for three ranges of pulse widths as shown in table 5.1.

Table 5.1: SET FIT (per Mcell) by Category and Pulse Width

Cell Type	$< 50ps$	$50ps..75ps$	$> 75ps$
INV/BUF	19	5	1
AND/NAND	42	15	3
OR/NOR	33	12	5
XOR/XNOR	109	59	6
Complex	147	65	39

In the second implementation of the methodology, the aggregation was per-



(a) Gate1 (b) Gate2

Figure 5.4: SET Sensitivity of Two 28nm Combinatorial Gates

formed both on gate function and on drive-strength. It was observed that the SET sensitivity is highly dependent on drive-strength as shown in figure 5.6.

5.3 Block Level LDR Modeling

The LDR for a single gate or for a full network depends on the state of the inputs. In this methodology, a set of pre-defined logic blocks are identified. An accelerated logic fault injection algorithm [Alexandrescu 2012] is used to compute the LDR of each gate in the circuit across the input vector space. For small circuits, this simulation can be exhaustive and for larger circuits a sufficient number of input vectors can be evaluated to calculate the LDR with a pre-defined level of accuracy [Nguyen 2005]. The result of the fault simulation is an average LDR value for each gate across the space of input vectors as well as a frequency distribution for each LDR value.

As part of these fault-injection simulations, the average *error propagation probability* through the circuit (from inputs to outputs) is calculated across all input vectors. Results are presented for several combinatorial arithmetic and data-path circuits in table 5.2. As expected, circuits consisting only of XOR gates (ripple carry adder, ECC encoder) have an internal LDR of *1.0*. The *error propagation probability* from inputs to outputs is independent on the architecture.

Figure 5.7(a) shows the result of the exhaustive LDR analysis on a 16-bit carry-look-ahead adder. We see that across the entire space of the input vectors, the average LDR of the block is *0.759* with a distribution of sensitivities between approximately *0.6* and *0.9*.

In real applications, the input vector distribution is clearly not uniform, but when performing a very early SER analysis, simulation traces are rarely available. Instead of just calculating the average LDR value for the circuits, the full histogram is recorded. Working with the average LDR gives a most likely estimate, however, the worst-case LDR could be used to provide an upper bound.

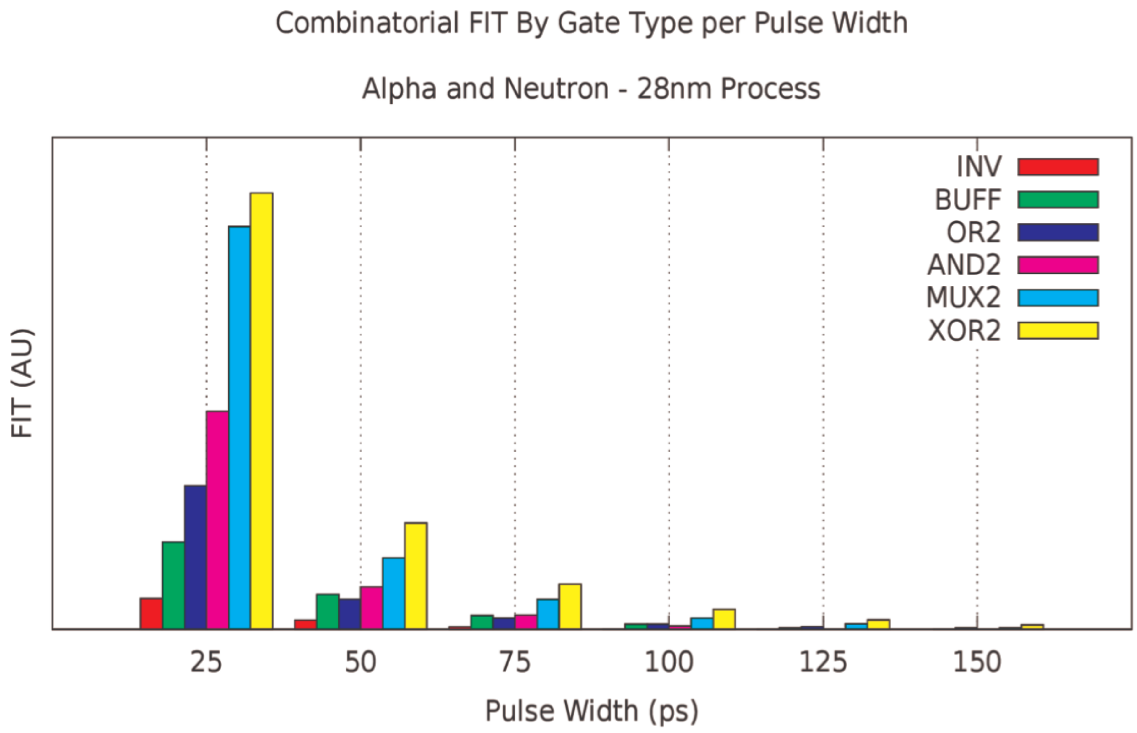


Figure 5.5: SET Sensitivity of Multiple 28nm Combinatorial Gates

5.4 Flat SET Analysis of Complex Circuit

5.4.1 Circuit Overview

To provide reference results against which to compare the proposed hierarchical SET analysis, the ALU from a 32-bit OpenRISC processor [OpenRisc 2013] was analyzed. The block-level structure of this ALU is shown in figure 5.8. The ALU performs standard addition, multiplication and shift operations. The RTL for this combinatorial circuit was placed between input and output flip-flops and synthesized flat using Synopys DC with a target frequency of 300 MHz. A SET fault injection campaign was then performed on the resulting net-list.

5.4.2 Flat SET Analysis of ALU Circuit

When generating the campaign, care was taken to ensure the distribution of faults and pulse-widths respected the relative likelihood of SETs in each gate. This was done by summing the FIT rates for all gate instances in the circuit, for all pulse-widths. Uniform random values were then selected over this range and then mapped back to the corresponding gate and pulse-width. The algorithm for selecting the target-gates and injected pulse-widths is shown in algorithm 1.

It is important to differentiate this approach with other work (such as [Miskov-Zivanov 2010]),

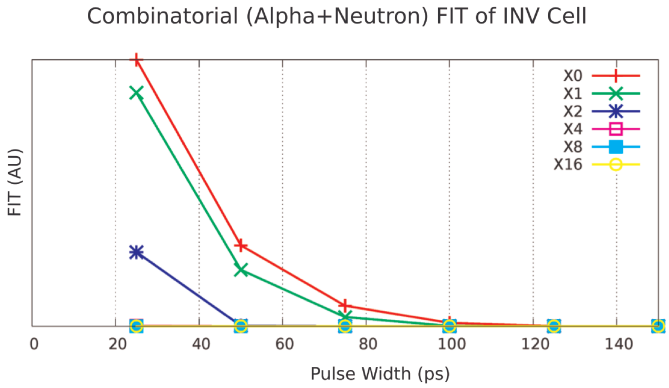
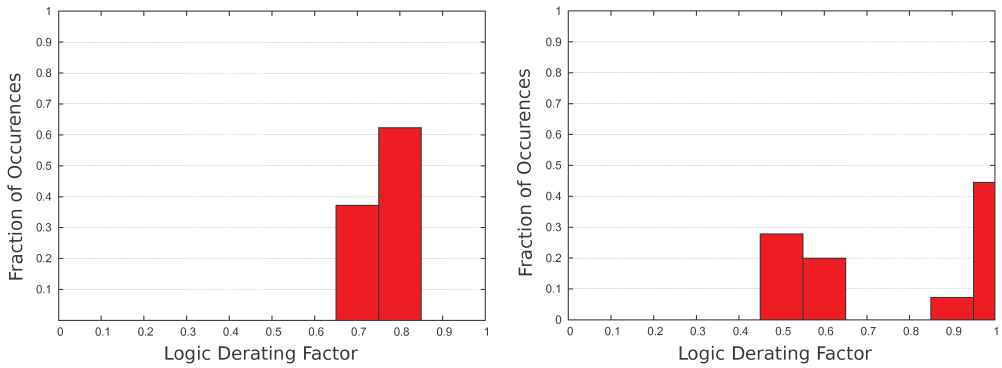


Figure 5.6: SET Sensitivity of Multiple 28nm Combinatorial Gates



(a) 16-bit CLA Adder (b) 64-bit ECC Decoder

Figure 5.7: LDR Histogram for Two Combinatorial Networks

which treats the pulse-width independently of the gate types. As seen in section 5.2, different gate types have different pulse-width distributions, thus this effect can not be ignored.

After the fault list was generated, a fault injection campaign was run using three types of gate-level simulation. The campaign was first run on a net-list with no delays and no setup-hold checks. An error was signaled if there was a mismatch in one or more output flip-flops. In this case, the effects of LDR and TDR were present, but there was no EDR and setup-hold violations are ignored. The same fault-list was then simulated with setup-hold checks enabled, but still no gate delays. Finally, the campaign was run with annotated gate delays. Inertial delays were enabled, so faults shorter than the propagation delay of a gate are filtered by the simulator, in order to provide a rudimentary EDR model.

The results for the three sets of runs are shown in table 5.3. The number of actual errors goes down when setup-hold checks are enabled. This is because temporal cases (c) and (e) from figure 1.9 result in timing violations rather than sampling of the

```
Data: FIT_Table[G, PW]
Compute Cumulative FIT
totalFit ⟵ 0 /* Compute Cumulative FIT                                  */
foreach G in GateInstances do
    foreach PW in PulseWidths do
        totalFit ⟵ totalfit + FIT_TABLE[G, PW]
    end
    CumFitPerGate[G] ⟵ totalFit
end
/* Perform Fault Injection                                              */
while ErrorInterval ≥ Threshold do
    /* Pick uniformly over all cases                                    */
    R ⟵ URandom(0..totalFit)
    G ← 0 /* Lookup Gate                                                */
    while R > CumFitPerGate[G] do
        G ← G + 1
    end
    R ⟵ R − CumFitPerGate[G]
    PW ⟵ minPW /* Lookup PW                                            */
    while R > FIT[G, PW] do
        R ⟵ R − FIT_TABLE[G, PW]
        PW ⟵ PW + PWStepSize
    end
    Offset ⟵ Urandom(0..T_clk) /* In cycle                             */
    Result ⟵ FaultInject(G, PW, TempOffset)
end
```

Algorithm 1: Weighted SET Fault Injection

Table 5.2: Gate Count and LDR for Combinatorial Blocks

Block	Cell Count	Internal LDR	Error Propagation
Ripple adder 8 bit	8	1.000	1.000
Rippler adder 16 bit	16	1.000	1.000
CLA adder 8 bit	41	0.878	1.000
CLA adder 16 bit	125	0.759	1.000
DX Multiplier 8 bit	387	0.928	0.960
DX multiply 16 bit	971	0.939	0.994
Equality comparator 16 bit		0.02	0.020
Equality comparator 32 bit		0.090	0.0632
ECC encoder 32 bit	61	1.000	1.000
ECC decoder 32 bit	165	0.778	0.921
ECC encoder 64 bit	128	1.000	1.000
ECC decoder 64 bit	308	0.772	0.949
MUX 8 to 1	7	0.748	0.135
MUX 16 to 1	21	0.705	0.069
MUX 64 to 1		0.621	0.018
Control Register Logic		0.478	0.260

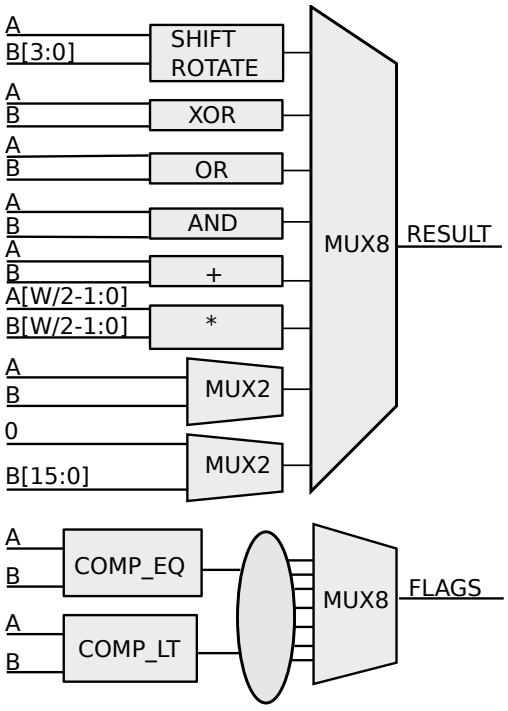


Figure 5.8: Block Diagram of ALU Circuit

incorrect value.

The total raw FIT rate of the gates was calculated to be *0.2173 FIT* and then de-rated using the results of the fault injection campaign, shown in the column *De-rated FIT*.

Table 5.3: Simulation Results for Flat Injection Campaign

Simulation Accuracy	Num. Runs	Num. Errors	Setup/ Hold	Not Blocked	PW Ratio	De-Rated FIT
Zero Delay	25 000	75	0	3272	1.0	0.65e-3
Zero Delay, S&H Checks	25 000	28	85	3272	1.0	0.61e-3
Delays, S&H Checks	25 000	45	77	2546	1.5	0.73e-3

Using the observability of the simulator, the D-input to the output flops was monitored to see how many of the faults actually reached the output of the combinatorial network (after LDR and EDR) and this is reported in the column 'Not Blocked'. The pulses that are blocked in the runs without timing delays are blocked solely due to LDR. The additional pulses that are blocked with the SDF simulations are the result of EDR filtering of short pulses. Based on digital simulation, the average or statistical EDR for this circuit is approximately $2546/3272 = 0.78$.

The width of the pulses exiting the combinatorial network could be longer or shorter than the original pulse, due to the reconvergence effects in the combinatorial logic. The column 'PW Ratio' shows the average ratio of the output pulse width to the original error pulse width. For those pulses that are not masked, on average the output pulse is about 50% longer.

5.5 Hierarchical Analysis of the Complex Circuit

In this section, we show how the effective SET FIT rate of the ALU circuit can be computed directly from the RTL using the methodology shown in figure 5.9. First the ALU circuit is decomposed into small combinatorial blocks, of the type that can be directly inferred by an RTL synthesis engine. Each of these blocks was synthesized using a generic (GTECH) library and the gate count and distribution of gate types obtained. The internal LDR of each of the small blocks was computed using fault injection and the resulting models for each block are shown in table 5.4. The generation of these models for standard logic blocks is a task that can be done in advance, as it is technology independent.

Combining the gate count and type distribution for each block with the results of the SET characterization of the target library (see table 5.1), an estimate of the *raw* intrinsic FIT rate for each of the small blocks was calculated, for different pulse widths. This is shown in the columns labelled 'Raw Intrinsic FIT' in table 5.4.

The raw intrinsic FIT of each block must be de-rated for LDR and TDR. The instrinc LDR has been pre-computed and is shown in the table. The SET intrinsic sensitivity has been pre-computed for different pulse widths (<50ps,50..75ps,>75ps) thus equation 1.3 can be applied as a discrete sum for each block to compute the de-rated, intrinsic FIT for each sub-block.

Now we must address the question of how the combinatorial SET models for the small networks can be combined to estimate the SER of the entire ALU. Consider a more general network as shown in figure 5.10.

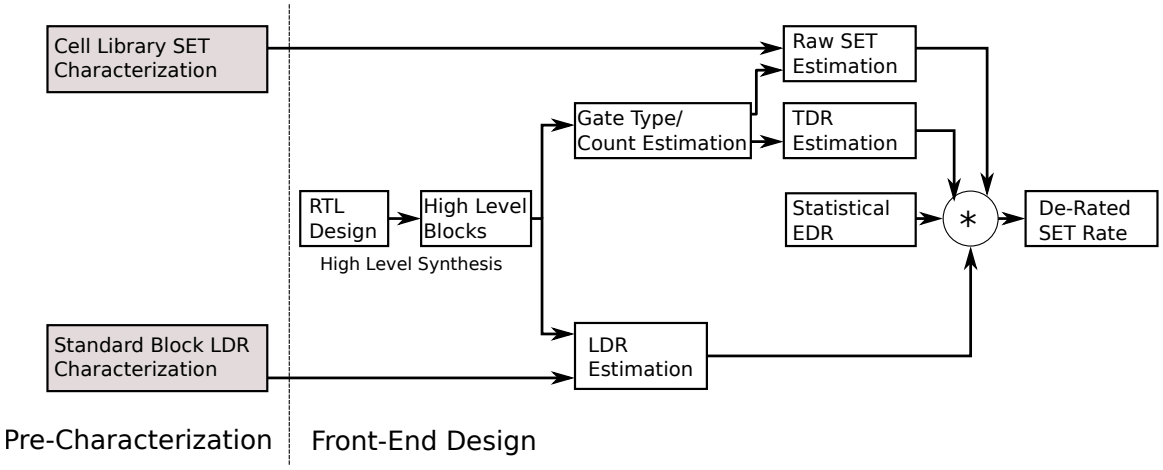


Figure 5.9: Methodology for SET Estimation from RTL

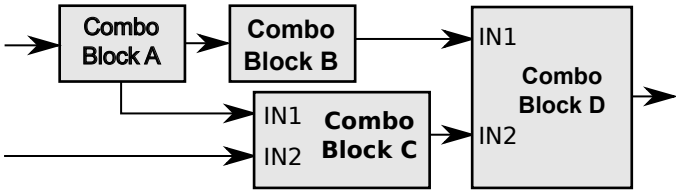


Figure 5.10: Complex Combinatorial Network

Each block has its own de-rated, intrinsic SET contribution. The SETs in the final block (D) propagate directly to the outputs. The SETs in the upstream blocks may be logically masked, thus their internal contribution is reduced by the error propagation probability (EPP) along the path to the output. The EPP is computed for each path to the outputs and the resulting propagation model is shown in equation 5.2. Since the computed LDRs are averages over many input vectors, the correlation of vectors on reconvergent paths need not be considered.

$$\begin{aligned} SER_{Network} = &(IntSER_A * EPP_B * EPP_{D1}) + \\ &(IntSER_A * EPP_{C1} * EPP_{D2}) + \\ &(IntSER_B * EPP_{D1}) + \\ &(IntSER_C * EPP_{D2}) + \\ &(IntSER_D) \end{aligned} \tag{5.2}$$

This approach was applied to the example ALU where the propagation probabilities of the branches are not equal due to the encoding of the op-codes which control the mux. The final estimated SET FIT rate for the ALU circuit, working from the RTL, is computed to be **0.76e-3** FIT which is only 15% higher than the value of **0.65e-3** FIT obtained from a fault-injection campaign on the flat-circuit without timing annotation as shown in table 5.3.

Table 5.4: Results of Block Level SET Characterization

Block Name	Gate Count	INV BUF	AND	OR	XOR	CMPLX	Intrinsic LDR	Raw Intrinsic FIT <50ps	50..75ps	>75ps
Bitwise AND	32	0	32	0	0	0	1.0	1.3e-3	4.6e-4	8.1e-5
Bitwise OR	32	0	0	0	0	0	1.0	1.0e-3	3.5e-4	1.4e-4
Bitwise XOR	32	0	0	0	32	0	1.0	3.3e-3	1.8e-3	1.7e-4
EQ Compare	46	1	12	1	30	2	0.02	3.9e-3	2.0e-3	2.7e-4
LT Compare	96	47	1	14	0	34	0.43	6.1e-3	2.5e-3	1.4e-3
Mux 1	32	0	0	0	0	32	1.0	4.5e-3	2.0e-3	1.2e-3
Mux 2	32	0	0	0	0	32	1.0	4.5e-3	2.0e-3	1.2e-3
Adder	204	6	38	70	70	58	0.86	1.9e-2	8.9e-3	2.9e-3
Multiplier	496	0	256	0	0	240	0.98	4.4e-2	1.8e-2	9.6e-3
Shift Left	160	0	32	0	0	129	0.70	1.9e-2	8.4e-3	4.8e-3
Shift Right	196	6	69	11	0	107	0.66	1.8e-2	7.8e-3	4.2e-3
Output Mux	182	5	42	2	0	133	0.72	2.0e-2	8.9e-3	5.1e-3

Currently, the methodology does not model the pulse deformation due to EDR. It is possible to apply the EDR ratio and pulse stretching ratio obtained from the gate-level simulations to the estimated FIT rate : 0.76e-3 * 0.78 * 1.5 = 0.9e-3 which is slightly higher than the results obtained from flat simulation with SDF (0.73e-3). This is only valuable if it is expected the the EDR effects do not vary across circuits.

It may appear that the absolute FIT rate of the example circuit is extremely small and one might assume the effect of SETs is not significant. However, the ALU being studied has approximately 1800 gates. If we scale the computed SET FIT rate of 0.76e-3 FIT up to a circuit with 100M gates and reduce the TDR effect by 3.3x, based on an assumed operating frequency of 1 GHz instead of 300 MHz, the absolute FIT rate would become approximately 140 FIT. This value is not negligible and would definitely need to be carefully considered in a SER analysis. Note, that even after a SET is captured in one or more flip-flops, it is still subject to the de-rating factors that affect SEUs and thus this value must be further de-rated.

5.6 Conclusions

In this chapter, a hierarchical methodology to estimate the effect of SETs was presented. The approach started with a simulation-based characterization of the sensitivity of a cell-library. This was distilled into a compact summary. An approach to pre-characterize the LDR of common building-block circuits was presented. The LDR is not technology dependent and can thus be computed in advance. A methodology to decompose a large circuit into smaller blocks, to compute the SER of each block and then to combine the results was presented. When applied to the OpenRISC ALU, this methodology showed good correlation with the results obtained using fault-injection on a flat net-list.

As presented, the approach does not consider the impact of input vectors. Input vectors do play a role. The challenge is that the number of input states grows exponentially with the number of inputs. One practical approach to reduce the complexity would be to focus on the impact of control signals. In the example ALU circuit, it would have been relatively easy to extend the methodology to compute the SER sensitivity for any ALU operation. Thus, instead of a single SER response, there would be one per-operation or eight in the case of this circuit. This remains a tractable number, however, to make use of a more refined model, requires the ability to write-out the model and to propagate it upwards to the next level of design hiearchy. This is where RIIF models can play a role.

As presented, the methodology does not propose a predective EDR model. Of course this is important, but it is not clear to what level of accuracy EDR can be predicted at the RTL level.

Chapter 6

Reliability Modeling with RIIF

Contents

6.1 Introduction

As seen in the introductory chapters, there exists an enormous body of work devoted to analyzing radiation effects and the propagation of faults to errors. However, there remains a significant gap between the state of the art in the research community and the best practices in industry, especially in terrestrial applications. One of the hurdles is the fact that there exists no framework for integrating the data about the rates of faults and de-rating factors. As a result, spreadsheets are commonly used to exchange such information [Wong 2012]. For example, foundries and library providers often use spreadsheets to communicate the FIT rates of memory and logic cells. Fault-injection simulations are frequently used to evaluate the sensitivity of a design to faults and the results of such analysis is often archived in the form of spreadsheets [Arlat 2011]. Unfortunately, every spreadsheet is unique and combining data from multiple spreadsheets is a manual and error-prone process.

In this chapter, we review the propagation of low-level faults upwards through a system hierarchy and in the process, we identify the information that is required to

model faults and how they propagate. The environment must be considered, as the operating conditions of a circuit (e.g. voltage, temperature, workload, etc.) play a key role in the determining the rate of occurence of faults as well as how the resulting errors eventually impact the system. The *Domain-Specific Language* [Fowler 2011] called Reliability Information Interchange Format (RIIF) is then presented. Three worked examples of systems modeled with RIIF are then presented.

The goals of the RIIF language are to:

- Enumerate the failure modes for components
- Specify the probability of the failure modes
- Specify the effect of operating parameters (e.g. voltage)
- Build composite components from simpler components
- Scale from cell level through to system-level
- Remain general purpose and not be tied to a single application domain
- Provide templates which standardize the specification of failures in commodity components (e.g. DRAMs, SRAMs).
- Specify standard operating conditions and environments for components

In industry, this type of reliability model has numerous direct applications, some examples of which are enumerated here.

1. When a component supplier provides a memory (e.g. SRAM, DRAM, TCAM) to a system company, they must deliver information about the soft error rates (e.g. SBU, MBU, SEFI, etc.). The rate of these events varies with parameters such as voltage, temperature, neutron flux and packaging options. It is beneficial for the system company to receive this information in a standard format so that they can quickly evaluate the reliability impact of selecting functionally equivalent components from different suppliers.

2. Cell library providers run simulations (e.g. SPICE or TCAD) and perform accelerated tests in order to determine the soft error susceptibility of the cells in their library. The number of cells in libraries is growing. For example, a frequently used commercial 28nm library contains hundreds of sequential cells and thousands of combinatorial cells. This information must be communicated in a machine readable format so that designers can compute the soft error rate of their circuits. Soft error rates vary significantly with voltage and the alpha contribution depends on the packaging options. There is also a strong correlation with voltage and a weaker correlation with process. To illustrate the importance of accurately characterizing each cell in a library, the simulated FIT rate (alpha plus neutron) of fourty different flip-flops from a commercial library are presented in figure 6.1. From this graph, it is clear that there is a large variation between the different cells in the same library. Furthermore, the SER of the master and slave stages can vary significantly

as shown by the results for $CK = 0$ and $CK = 1$. This need to perform accurate SER analysis which considers the SER characteristics of each cell was highlighted in [Alexandrescu 2013].

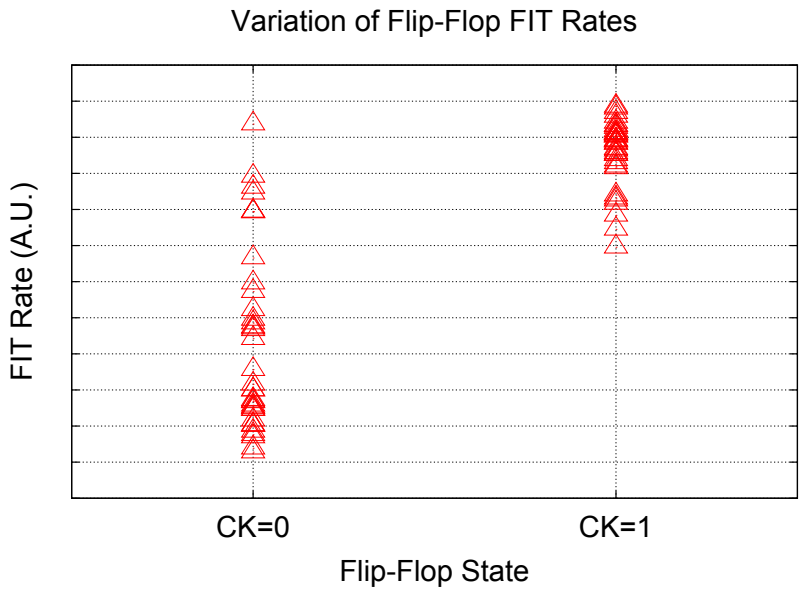


Figure 6.1: Variation of Flip-Flops SER for Cells from a 28nm Library

In many ways, the problem is analagous to that of specifying the timing characteristics of standard cells which also depend on process, temperature and voltage. In the case of timing information, standard file formats, such as the Liberty [Synopsys 2006], already exist.

3. When an IP provider delivers a complex block (e.g. processor core, Ethernet controller, etc.), there is a need for them to specify the failure modes and errors that can occur. This need for soft error models for IP blocks was identified in [Aitken 2005]. A simplified view of the design flow for an automotive IC is shown in figure 6.2 and highlights the complexity of the industrial relationships. The SoC integrator needs to show compliance with reliability standards even though a large fraction of the IC design content comes from third parties. Therefore, they must have the means to produce an accurate reliability model for the SoC.

For example, if an IP block has parity protected structures, then there is a DUE failure mode whose rate of occurence depends on the number of bits that are protected and the LDR of the structure, both of which are technology independent. When this IP is mapped to a given technology, this information can be combined with the library soft error models to compute an absolute FIT rate. An example of this flow is presented in section 6.3.2.

4. When a system company designs a board which has multiple chips (as well as other components), there is a need to develop a comprehensive failure model for the board which includes both intermittent and permanent failure modes. Certain

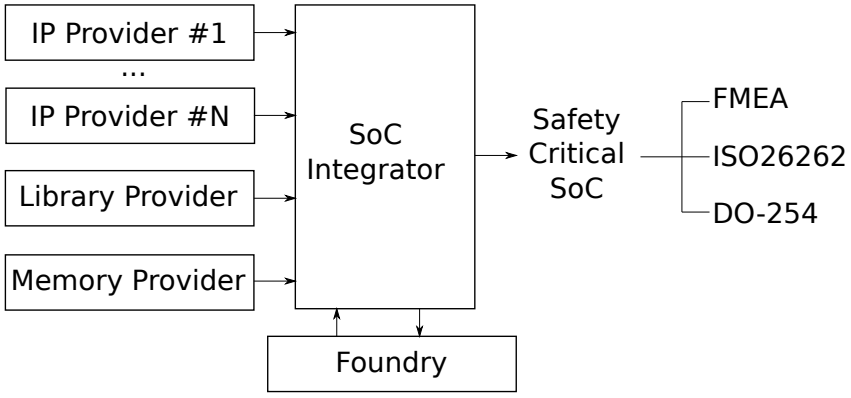


Figure 6.2: Simplified Design Flow of an Automotive SoC

component level failure modes may not impact system reliability due to redundancy or error correction - so it is not possible to simply sum the FIT rates. There are different ways in which the board can fail (e.g. reboot, performance degradation, permanent failure), and it is necessary to individually compute the probability of each failure mode. It is important to preserve the relationship with the underlying component models so that the impact of changes to operating parameters (e.g. voltages and temperature) can be assessed. Although, there do exist industrial standards for evaluating the reliability of circuit cards [Telecordia 2011], they do not take into account fault mitigation techniques and they use a very simple model of failures in complex ICs based on a single FIT rate computed from the number of transistors. As a result, the Mean Time Between Failure (MTBF) values predicted by these standard techniques are often unnecessarily pessimistic.

6.2 Elements of the RIIF Language

6.2.1 Components and Parameters

The basic unit of enapsulation in RIIF is the *component* which is similar to a VHDL entity or a Verilog *module* and it can represent either a low-level circuit such as a logic gate or a complex entity such as a full SoC. Of course, a given circuit can operate under different conditions, and thus *components* have *parameters* which are similar to *generics* in VHDL. The *parameters* are typed (integer, float) and they may optionally have a value assigned when they are declared. The value of a parameter can be over-ridden when the component is used. RIIF also supports *constants* which are like *parameters*, with the difference that a value must be assigned when the *constant* is declared and this value can not be changed.

6.2.2 Failure Modes

With RIIF, the intent is to only model the faults in the system but not to model its normal functionality. The user declares *failure_ modes* within the *components*. A *failure_ mode* represents an event that can occur and the purpose of RIIF is to calculate the rate of these events. In its current form, the language does not make the distinction between faults, errors and failures. The lack of such a distinction comes from the fact that the goal of the language is to be generic and span multiple layers of abstraction. A modification to the language to make this distinction explicit, is described in section 6.4.

Examples of common *failure modes* in a memory *component* would be a SBUs while in a CPU, SDC would be a *failure_ mode*. Associated with each *failure mode* is a rate of occurence. Figure 6.3 shows examples of failure modes at different levels in a silicon system.

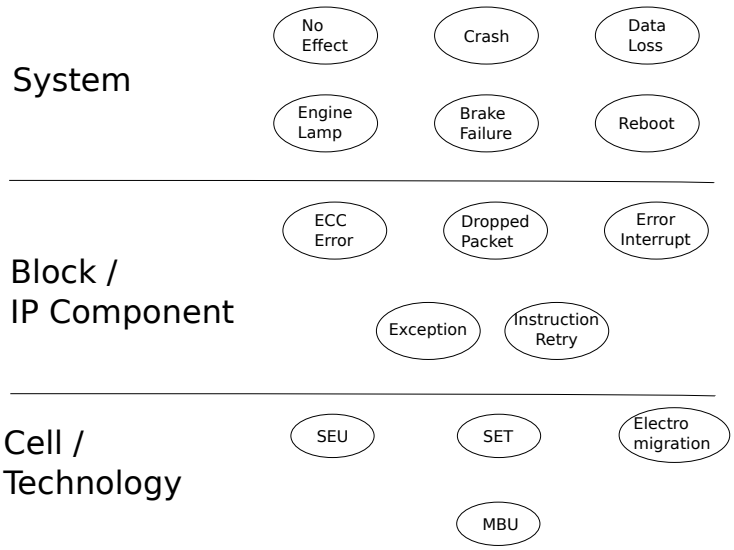


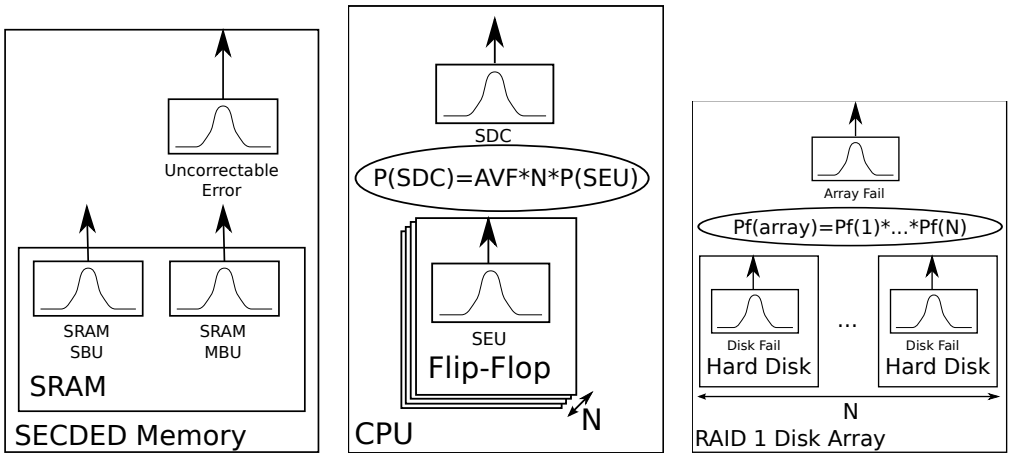
Figure 6.3: Failure Modes at Different Levels of Silicon Systems

6.2.3 Complex Components

A complex *component* is built from one or more *child components*. The complex *component* has its own failure modes, whose rate of occurence is expressed as a function of the rates of the failure modes in the *child components* as well as the *parameters* in the complex *component*. The function which maps the rate of lower level *failure modes* to higher level *failure modes* can take into account the effects of error mitigation, error masking and redundancy as shown by the three examples which follow.

In figure 6.4(a), a SRAM is the *child component* of a Single Error Correct,

Double Error Detect (SECDED) memory. In this case, the SBU faults from the SRAM do not propagate and the MBU faults map directly to uncorrectable errors in the SECDED memory. In figure 6.4(b), a simple example of a model for SDC in a CPU is shown. The sum of the SEU rate in the N flip-flops is scaled by a single AVF factor to obtain the rate of SDC.



(a) Error Mitigation (b) Scaling of Fault Rates (c) Modeling Redundancy

Figure 6.4: Computing Rates of Failure Modes in Complex Components

Redundancy is often used to improve reliability and availability. In figure 6.4(c), the structure of a RIIF model for a RAID-1 disk array is shown. In this system, data is replicated across the N drives. As long as one of the drives is operational, the data can be retrieved, thus the probability of the array failing is the product of the rate of failure of each disk. The reliability of series, parallel and combined series parallel systems has been extensively studied. Well known, closed form expressions exist for simple systems and techniques to obtain bounds for complex systems have also been developed [Gertsbakh 1989, Chapter 1]. In RIIF, there are built-in functions (*agg_single_fail*, *agg_gt_n_fail*) which compute these relationships. The RIIF code for the RAID example is shown in figure 6.5.

6.2.4 Environments

The operating environment of an integrated circuit greatly impacts its reliability. The fact that RIIF *components* are parameterized provides a first step towards managing this dependence. However, maintaining models with a long list of parameters is not scalable. Furthermore, if different users define different sets of *parameters* for the same class of device, then they are no longer modular. There is thus a need to encapsulate a set of related *parameters* to define an *environment*. Essentially, an *environment* is similar to a 'C' struct and serves to encapsulate a set of *parameters*. Going forward, RIIF implementations of standard environments can be defined,

```
component RAID_1_ARRAY;
   parameter N : int := 2; // number of disks

   child_component HARD_DISK[1:N]; // instantiate child components

   fail_mode : ARRAY_FAIL; // failure mode in the array

   // Array only fails if greater than (N−1) disks fail
   assign ARRAY_FAIL'rate = agg_gt_n_fail( (N−1), HARD_DISK[1:N]'DISK_FAIL ) ;

endcomponent
```

Figure 6.5: RIIF Code for RAID-1 Disk Array

standardized and used across multiple models.

```
environment NEUTRON_ENV;
   input parameter LOCATION : enum { NYC, LOS_ALAMOS, TOKYO } := NYC;
   input parameter SOLAR_ACTIVITY : enum { LOW, PEAK, AVG } := AVG;

   output parameter REL_FLUX : float;

   assign REL_FLUX = ( LOCATION == NYC ) ?
                        ( ( SOLAR_ACTIVITY == AVG ) ? 1.0 :
                          ( SOLAR_ACTIVITY == LOW ) ? 0.927 : 1.073 ) :
                     ( LOCATION == LOS_ALAMOS ) ?
                        ( ( SOLAR_ACTIVITY == AVG ) ? 5.6 :
                          ( SOLAR_ACTIVITY == LOW ) ? 5.15 : 5.70 ) :
                     ( LOCATION == TOKYO ) ?
                        ( ( SOLAR_ACTIVITY == AVG ) ? 0.66 :
                          ( SOLAR_ACTIVITY == LOW ) ? 0.62: 0.64 ) : NaN ;

endenvironment
```

Figure 6.6: Example of a RIIF Environment

In figure 6.6, we present the *RIIF* code for an *environment* construct that captures a small subset of the neutron model presented in [JEDEC 2006]. An *environment* construct has a list of *parameters* (e.g. location) and it provides a list of generated or output *parameters* (e.g. neutron flux). Typically, an *environment* construct is used to encapsulate the operating conditions in a specific product (e.g. temperature, voltage range in an automotive application) and then to consistently apply these parameters to a set of components.

6.3 Worked Examples

6.3.1 Board Level Example

The first example developed with RIIF was based on a circuit board with one CPU and with $N + 1$ memory components. We assume that the data in each RAM is

protected with SECDED ECC so that single-bit errors have no impact. When there is an intermittent multi-bit error (MBE), the board is rebooted, but continues operation. Due to $N+1 : N$ redundancy on the memory parts, if there is a permanent failure on one RAM, the board can continue to operate with N components, although the switch-over requires a reboot. If two or more RAMs fail, then the board must be replaced. The structure of the system is shown in figure 6.7.

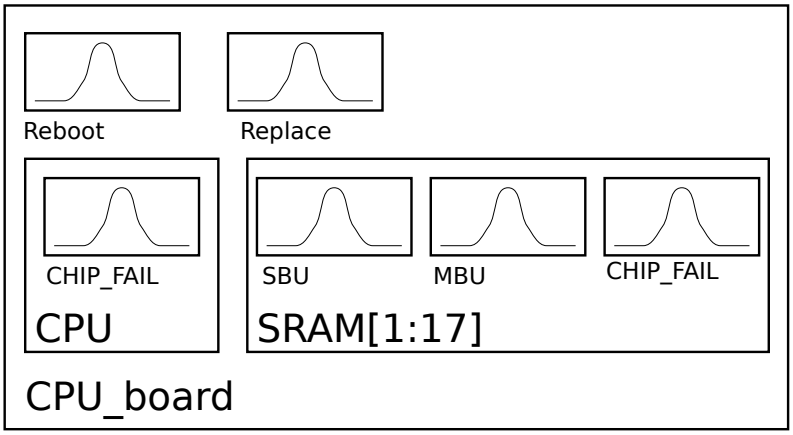


Figure 6.7: Structure of Board with CPU and RAMs

First, the RIIF code for the SRAM is presented in figure 6.8. In this model, the SRAM has three failure modes : *SBU*, *MBU* and *CHIP_FAIL*. The equation for the SBU rates is taken from [Seifert 2006] and the MBU rate is computed as a fixed ratio of the SBUs. Then, the rate of permanent chip failues is calculated as a function of the operating temperature. It is important to note, that the purpose of RIIF is not to propose specific models for silicon failure mechanisms. This topic has been extensively studied and standard equations exist for known silicon failure mechanisms [JEDEC 2010]. Instead the purpose of RIIF, is to provide a scalable, machine-readable format for such models.

Next the model for the board is created, as shown in figure 6.9. The *failure modes* at the board level are reboots and board replacements. The board is rebooted if there is an MBU in one of the active memories or if a permanent memory error causes a switch-over to the spare memory part. The board must be replaced if either the CPU fails or if more than one memory part fails.

```
component SRAM;
 // Design parameters (geometry)
 parameter WIDTH : int := 32; // word width
 parameter M : int := 1024; // number of words
 parameter SIZE: int := M * WIDTH;
 // Parameters related to the environment
 parameter TEMP : float; // kelvin
 parameter VOLTAGE : float;
 // Technology Constants
 constant Ea : float := 0.6; // activation energy
 constant BASE_TEMP : float := 298; // base temperature for hard FIT rate (K degree)
 constant BASE_FAIL : float := 5; // hard-FIT rate at base temperature
 constant A_DIFF : float := 3.2; // constant obtained from layout
 constant Q_COL_EFF : float := 0.6; // represent Qcrit/Qcoll
 constant MBU_RATIO : float := 0.25; // Simplified model for MBUs

 // Failure Modes
 fail_mode SBU; // SBU obtained from (Seifert 2006)
 assign SBU'unit = FIT;
 assign SBU'rate = SIZE *
                   NEUTRON_ENV.RELATIVE_FLUX *
                   A_DIFF * EXP( -VOLTAGE / Q_COLL_EFF );
 assign MBU'rate = SBU'rate * MBU_RATIO;

 fail_mode CHIP_FAIL; // Failure rate scaled based on Arrhenius equation
 assign CHIP_FAIL'unit = FIT;
 assign CHIP_FAIL'rate = BASE_FAIL * exp( ( Ea / k ) * ( 1/TEMP - 1/BASE_TEMP) ) ;
endcomponent // SRAM
```

Figure 6.8: RIIF Model for a Generic SRAM

```
component CPU_BOARD;
 // ------------------ Constant Declaration ------------------------
 constant NUM_RAMS : integer := 17;
 parameter CHIP_VOLTAGE : float := 1.0;
 constant MEMORY_UTILIZATION : float := 0.75; // Used for memory de-rating
 // ------------------ Instantiate Components ----------------------
 child_component CPU CPU0; // Single CPU
 assign CPU0.VOLTAGE = CHIP_VOLTAGE; // Propagate parameters to children
 child_component SIMPLE_SRAM MEM[1:NUM_RAMS]; // 17 memory chips
 assign MEM[1..NUM_RAMS].VOLTAGE = CHIP_VOLTAGE;
 // ------------------- Define Failure Modes for the Board ----------------------
 // If there is an MBU in any memory or if the card must fail-over on a permanent RAM failure.
 fail_mode REBOOT;
 assign REBOOT'rate = MEMORY_UTILIZATION *
 agg_single_fail( MEM[1:NUM_RAMS]'MBU ) + agg_single_fail( MEM[1:(NUM_RAMS-1)]'CHIP_FAIL );

 // Board is replaced if more than 1 memory chip fails or if CPU fails
 fail_mode REPLACE_BOARD;
 assign REPLACE_BOARD'rate = agg_single_fail( CPU'CHIP_FAIL,
                                              agg_gt_n_fail( 1, MEM[1:NUM_RAMS]'CHIP_FAIL ) );

endcomponent // CPU_BOARD
```

Figure 6.9: RIIF Model CPU Board

6.3.2 Soft IP Example

The example in this section illustrates how a model for a soft processor core can be specified. The high-level failure modes of interest for this core are SDC and DUE. The SoC integrator is interested in determining the absolute FIT rate for SDC and DUE in their implementation technology. These FIT rates can then be fed into a system level reliability or safety analysis.

The SoC integrator, through their relationship with the foundry and the memory IP provider, has access to the technology failure rate (e.g. FIT/Mbit for memories, flip-flops) but they lack detailed knowledge of the processor core architecture. We show how the IP provider can deliver a technology-independent RIIF model of the core. This model is then extended with the models for the technology failure rates to compute the absolute FIT rates.

In this example, we assume there are four technology level faults :

- SBUs in the cache memory
- MBUs in the cache memory
- SEUs in the normal flip-flops
- SEUs in the hardened flip-flops

As is the case with many IPs, the core is configurable. The user can choose from different protection strategies for the cache SRAM : (i) none, (ii) parity or (iii) SECDED ECC. The selection is represented as a *parameter* in the RIIF model and the setting of these parameters influences how faults map to SDC or DUE. The model is illustrated in figure 6.10. On the bottom are the four technology level faults which feed into the model for the model for the CPU core.

It is assumed that the IP provider has performed fault-grading on the flip-flops and divided them into two sets : *critical* and *non-critical*. The AVF for SDC events has been determined for each set through techniques such as those discussed in chapter 3. The user of the IP may chose to map the *critical* set of flip-flops to a robust cell. For this reason, the model has separate *parameters* for the FIT of *critical* and *non-critical* flip-flops.

The rates of SDC and DUE can be calculated using the following equations:

$$FIT_{SDC} = Cache_{SDC} + FF_{SDC} \tag{6.1}$$

$$FIT_{DUE} = Cache_{DUE} + FF_{DUE} \tag{6.2}$$

$$Cache_{SDC} = N_{BITS} \cdot Cache_{Utilization} \cdot \begin{cases} FIT_{SBU} + FIT_{MBU} & \text{No Cache Protection} \\ FIT_{MBU} & \text{Parity Protection} \\ 0 & \text{SECDED ECC} \end{cases} \tag{6.3}$$

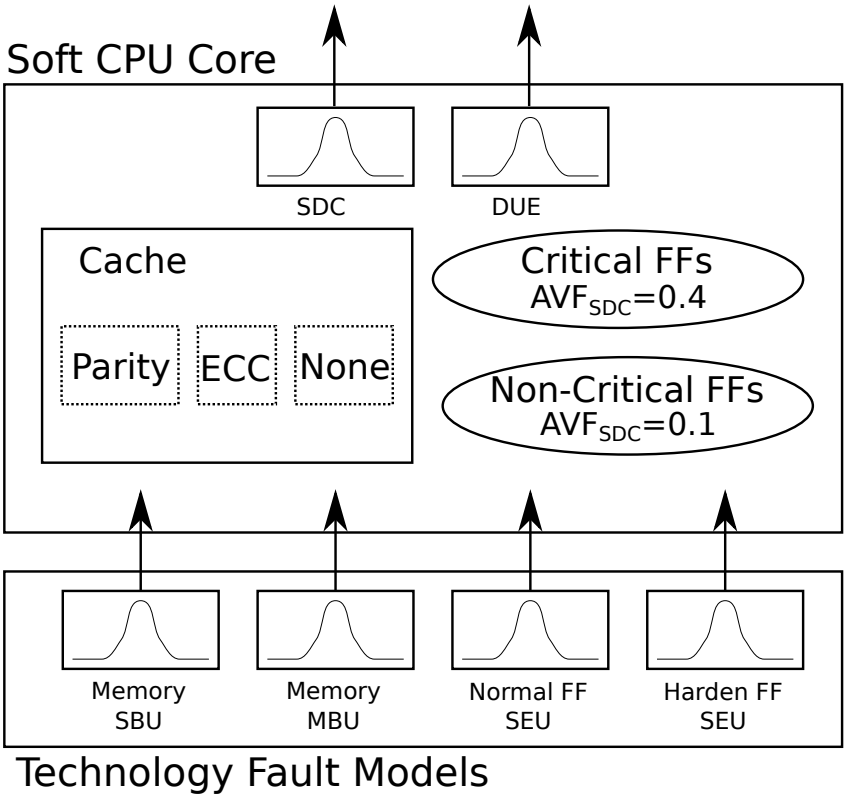


Figure 6.10: Fault Propagation Through a Soft CPU Core

$$Cache_{DUE} = \begin{cases} 0 & \text{No Cache Protection} \\ FIT_{SBU} & \text{Parity Protection} \\ FIT_{MBU} & \text{SECDED ECC} \end{cases} \tag{6.4}$$

$$FF_{SDC} = AVF_{CRIT\_FF} \cdot N_{CRIT\_FF} \cdot FIT_{CRIT\_FF} + AVF_{NORM\_FF} \cdot N_{NORM\_FF} \cdot FIT_{NORM\_FF} \tag{6.5}$$

$$FF_{DUE} = AVF_{DUE} \cdot (N_{CRIT\_FF} \cdot FIT_{CRIT\_FF} + N_{NORM\_FF} \cdot FIT_{NORM\_FF}) \tag{6.6}$$

The code to implement these equations as a component is presented in figure 6.11. There is a *component* declaration, followed by a set of *parameter* declarations. The first set of parameters specifies how the IP has been configured, specifically the cache protection scheme. Next, there is a parameter to represent the utilization of the cache. This is a MDR factor and can be set based on application profiles. The next four parameters are the FIT rates for the technology level faults and these have no values assigned to them.

The failure modes for the processor are defined and the equations above are used to calculate the SDC and DUE rates. The RIIF language is not procedural, thus the ternary operator (?:) is used to conditionally select which faults contribute to SDC or DUE based on the chosen cache protection scheme.

```
component SOFT_CPU_CORE;
 // Define Design Parameters
 parameter CACHE_EN : int := 1; // Cache enabled in application? (1=TRUE)
 parameter CACHE_ECC_EN : int := 1; // ECC enabled on the cache? (1=TRUE)
 parameter CACHE_PARITY_EN : int := 0; // Parity enabled on the cache? (1=TRUE)

 // Define Application Parameters
 parameter CACHE_UTIL : float := 0.5; // Cache utilization. Affects cache SBE/MBE rate

 // Declare Technology Failure Rate Parameters (Values assigned when model is mapped )
 parameter SRAM_SBU_FIT : float; // FIT/MBIT
 parameter SRAM_MBU_FIT : float; // FIT/MBIT
 parameter FF_NORM_FIT : float; // FIT/MBIT of FFs used for normal FFs
 parameter FF_CRIT_FIT : float; // FIT/MBIT of FFs used for critical FFs

 // Architectural Design Constants Pertaining to the CPU Core
 constant NUM_CACHE_LINES : int := 16*1024;
 constant NUM_BITS_CACHE : int := NUM_CACHE_LINES * ( 72 + 16 ); // data + TAGs
 constant NUM_CRIT_FFS : int := 8000; // based on AVF
 constant NUM_NON_CRIT_FFS : int := 50000; // based on AVF
 constant CRIT_FF_SDC_AVF : float := 0.4; // AVF of critical FFs.
 constant NON_CRIT_FF_SDC_AVF : float := 0.1; // AVF of non-critical FFs.
 constant ALL_FF_DUE_AVF : float := 0.3; // DUE AVF of FFs

 // Define Failure Modes
 fail_mode CACHE_CORR_ERR; // Correctable Cache Errors
 assign CACHE_CORR_ERR'rate = ( CACHE_EN && CACHE_ECC_EN ) ?
     ( NUM_BITS_CACHE * CACHE_UTIL * SRAM_SBU_FIT / 1000000 ) : 0;

 fail_mode CACHE_UNCORR_ERR; // Uncorrectable Cache Errors
 assign CACHE_UNCORR_ERR'rate = ( CACHE_EN && CACHE_PARITY_EN ) ?
       ( NUM_BITS_CACHE * CACHE_UTIL * SRAM_SBU_FIT / 1000000 ) :
   ( CACHE_EN && CACHE_ECC_EN ) ?
       ( NUM_BITS_CACHE * CACHE_UTIL * SRAM_MBU_FIT / 1000000 ) : 0;

 fail_mode CACHE_SDC; // Silent Data Corruption in the Cache
 assign CACHE_SDC'rate = ( CACHE_EN && !CACHE_PARITY_EN && !CACHE_ECC_EN ) ?
       ( NUM_BITS_CACHE * CACHE_UTIL * ( SRAM_SBU_FIT + SRAM_MBU_RATE ) / 1000000 ) :
   ( CACHE_EN && CACHE_PAR_EN ) ?
       ( NUM_BITS_CACHE * CACHE_UTIL * SRAM_MBU_FIT / 1000000 ) : 0 ;

 fail_mode DUE; // Detectable Uncorrectable Errors
 assign DUE'rate = ALL_FF_DUE_AVF *
   ( ( FF_NORM_FIT * NUM_NON_CRIT_FFS ) + ( FF_CRIT_FF * NUM_CRIT_FFS ) ) +
      CACHE_UNCORR_ERR'rate; // Cache uncorrectable produces DUE

 fail_mode SDC; // Silent Data Corruption
 assign SDC'rate = ( ( FF_NORM_FIT * NUM_NON_CRIT_FFS * NON_CRIT_FF_SDC_AVF ) +
                     ( FF_CRIT_FIT * NUM_CRIT_FFS * CRIT_FF_SDC_AVF ) ) +
                       CACHE_SDC'rate; // SDC from cache
endcomponent // SOFT_CPU_CORE
```

Figure 6.11: RIIF Model of a Soft CPU Core

6.3.2.1 Technology Mapping

The model presented in figure 6.11 is technology independent and must now be combined with technology models to compute absolute FIT rates. A second way of combining models in RIIF is through an Object Oriented (OO) approach [Meyer 1997]. A new component may be derived from an existing one by extending it, similar to other OO languages such as C++ or SystemVerilog. All of the elements (parameters, failure modes) in the original component are visible in the extended component[1].

A RIIF model for a generic SRAM was already seen in figure 6.8. This generic component can be extended to create a technology specific model, as shown in figure 6.12 where FIT rates are specified for a specific technology. For the purpose of this example, the SBU and MBU FIT rates are obtained by a lookup from assumed radiation test-results and coded using the ternary operator (?:). The full object inheritance hierarchy is shown in figure 6.14.

```
component SRAM_45NM extends SRAM;
 assign SBU'rate = ( VOLTAGE < 0.7 ) ? 400 :
                   ( VOLTAGE < 0.8 ) ? 350 : 300;
 fail_mode MBU;
 assign MBU'rate = ( VOLTAGE < 0.7 ) ? 35 :
                    ( VOLTAGE < 0.8 ) ? 20 : 10;
endcomponent // SRAM_45NM
```

Figure 6.12: RIIF Model for a Technology Specific SRAM

Finally, these models can be combined to yield a model for the soft core implemented in a specific technology. This is done using OO extension. In the compact model, shown in figure 6.13, the technology FIT rates are mapped to the parameters in the model of the soft-core.

```
component CPU_CORE_45NM extends SOFT_CPU_CORE, DICE_FF_45NM, STD_FF_45NM, SRAM_45M;
 parameter SOFT_CPU_CORE::SRAM_SBU_FIT = SRAM_45NM::SBU'rate;
 parameter SOFT_CPU_CORE::SRAM_MBU_FIT = SRAM_45NM::MBU'rate;
 parameter SOFT_CPU_CORE::FF_NORM_FIT = STD_FF_45NM::SEU'rate;
 parameter SOFT_CPU_CORE::FF_CRIT_FIT = DICE_FF_45NM::SEU'rate;
endcomponent // CPU_CORE_45NM
```

Figure 6.13: Combining CPU and SRAM Models

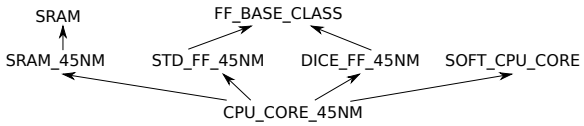


Figure 6.14: Object Inheritance Diagram for Soft CPU Core Example

[1]Currently RIIF does not have *public* or *private* data, but this is an obvious extension.

6.3.3 Scrubbed Memory Example

Correctly modeling the effective failure rate even of a simple system can quickly become complicated when multiple failure modes are considered and when mitigation techniques mask some of the errors. In [Sanchez-Macian 2013], the authors developed a RIIF model for an SRAM with M words. The model assumes that the memory is subject to transient SBUs at a rate of λ_{SBU} and to permanent Single Bit Hard Errors (SHEs) at a rate of λ_{SHE}. It is assumed that this memory is protected by a Single Error Correct (SEC) ECC. If two bits in a word are wrong (two SBUs, two SHEs or an SBU and an SHE), the result is Silent Data Corruption (SDC).

SBUs accumulate in the memory starting from the time the system is powered-up, t. Hard errors (SHEs) accumulate in the memory starting at the time the system was built, t'. [Sanchez-Macian 2013] show that the rate of data corruption with this system is:

$$\lambda_{SDC} = \frac{\lambda_{SBU}^2 \cdot t}{M} + \frac{\lambda_{SHE}^2 \cdot t'}{M} + \lambda_{SBU} \cdot \frac{\lambda_{SHE} \cdot t'}{M} + \lambda_{SHE} \cdot \frac{\lambda_{SBU} \cdot t}{M} \tag{6.7}$$

It is common practice to use scrubbing [Saleh 1990] to prevent error accumulation. Through scrubbing, the time period during which SBUs can accumulate is limited to T_s rather than the time since power-up, t. With scrubbing enabled, the rate of data corruption is given by:

$$\lambda_{SDC} = \frac{\lambda_{SBU}^2 \cdot T_S}{2 \cdot M} + \frac{\lambda_{SHE}^2 \cdot t'}{M} + \lambda_{SBU} \cdot \frac{\lambda_{SHE} \cdot t'}{M} + \lambda_{SHE} \cdot \frac{\lambda_{SBU} \cdot T_S}{M} \tag{6.8}$$

These equations can be coded in *RIIF* as shown in figure 6.16. First, the base *component* for the SRAM used in this example is shown in figure 6.15. This model is nearly identical to that in 6.12, except for the inclusion of the SHE failure mode. Clearly, it is undesirable to have inconsistent sets of failure modes for the same component, thus there is a need to establish, libraries of base components which will be standardized. The base model defines the rates of SBUs and SHEs as a function of voltage and temperature.

A new *compoment* is declared which extends the base *component* and which models the effects of ECC and scrubbing. The code for this extension is shown in figure 6.16. This model can be processed by the RIIF processor and the results of simulating the models have been plotted in figure 6.17. From the results, we see that with scrubbing, the SDC rate grows slowly with time, as the probability of an SBU and SHE occurring in the same word is small. Without scrubbing, of course, the SDC rate grows quickly with the time since power-up, as the SBUs accumulate in the memory (red/crosses, blue/star lines). With scrubbing enabled, the SDC rate due to SBUs is flat with time. However, due the the fact that the SHE rate increases with time, the total SDC rate is higher after ten years (green/Xs, purple/boxes lines) than during the first year.

This example shows that even a simple system requires a time-variant model that represents the effect of mitigation schemes such as scrubbing and ECC.

```
component SRAM; // SRAM with two failure modes
 // Define Parameters
 parameter SIZE: int := 512 * 1024 * 1024;
 parameter WIDTH : int := 32;
 parameter M = ( SIZE / WIDTH );
 // Operating Parameters
 parameter TEMPERATURE: float;
 parameter VOLTAGE : float;
 // Parameter associate with time
 parameter BUILD_T : time; // manufacturing time
 parameter POWERUP_T : time; // power-up time
 // Failure modes
 fail_mode : SBU; // soft error
 assign SBU'unit = FIT;

 fail_mode : SHE; // hard error
 assign SHE'unit = FIT;
endcomponent
```

Figure 6.15: RIIF Model for SRAM Model with SEC ECC and Scrubbing

```
component SEC_PROTECTED_SRAM extends SRAM;
 parameter T_SCRUB : time; // scrub interval
 assign T_SCRUB=express_time("hours",24);
 fail_mode SDC; // data corruption
 assign SDC'rate = ( pow(SBU'rate,2) * get_time_since( POWERUP_T, "hours" )/M +
                     pow(SHE'rate,2) * get_time_since (BUILD_T, "hours" )/M +
                    (SBU'rate * SHE'rate / M) * ( get_time_since( POWERUP_T, "hours" ) +
                                                  get_time_since( BUILD_T, "hours" ) ) );
endcomponent
```

Figure 6.16: RIIF Model for SRAM Model with SEC ECC and Scrubbing

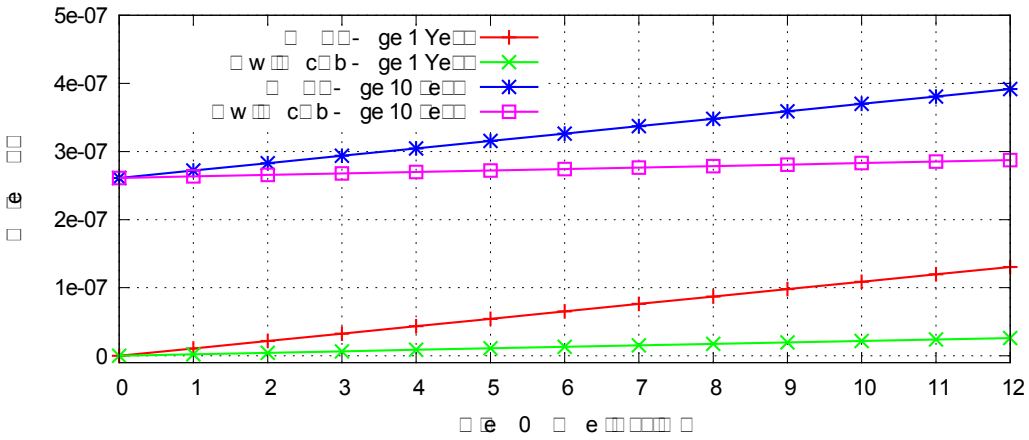


Figure 6.17: Results of Simulating SRAM RIIF Model

6.4 Extensions to RIIF

6.4.1 Faults versus Errors

In its current form, the RIIF language makes no distinction between faults, errors and failures. The notion of a *failure mode* is general and can cover any type of event that occurs. When the RIIF model is flattened, these events form a Directed Acyclical Graph (DAG), and the rate of the events is propagated from the leaf-level upwards. This approach is flexible, but may lead to ambiguity for the users. A more formal approach, shown in figure 6.18 has been considered where this distinction is explicit. A special *technology component* would be used to model faults. This type of component would define faults and their rate of occurence and would explicitely be at the bottom of the hierarchy.

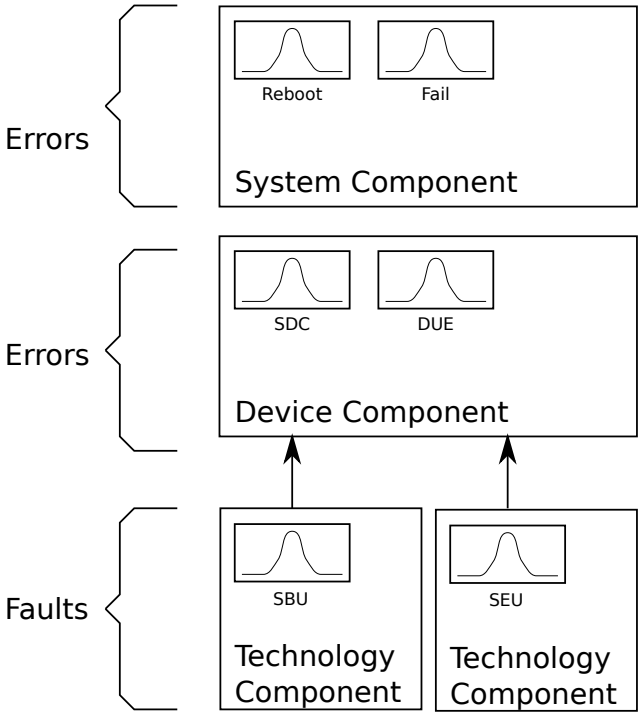


Figure 6.18: Formalizing the Difference Between Faults, Errors and Failures

Next in the hierarchy would be *device components*. Semantically, these would require as input, a set of technology components for the known faults. Based on these fault models, the *device component* would compute the rate of specific errors. Then, at the top of the hierarchy, *system components* would take as input specific *errors* from the *technology components* and calculate *failure rates*.

It seems the key distinctions with this approach are twofold. First, *technology components* only define faults but take no other input about incoming event rates. At elaboration time, it would be verified that *technology components* are at the leaf-level of the tree. Secondly, the *device* and *system* components would explicitely

declare which faults or errors are required as inputs from the level below. This would ensure there is no mismatch between the set of faults that exist, and those which are analyzed.

6.4.2 Mission Profiles and Timelines

Correctly modeling the environment and the mission profile of a system is critical in order to assess its reliability. Many systems do not operate continously with a homogenous workload. This is clearly the case of automotive or avionic systems where there are distinct states of operation. RIIF does not yet address this issue, although it is recognized as an important future contribution. In the following paragraphs, some early considerations about this aspect are presented.

Many system companies have extensive historical data on actual usage profiles. For example car manufacturers collect data on the number of hours and distances that are driven. For the purposes of reliability modeling, the goal is not necessarily to apply a specific historical profile, but rather to identify one or more *typical* profiles. One approach consists in using a probablistic model of the system state (Markov model). The duration of each state and the transition probabilities are determined based on actual profiles.

As a simple example, we consider an airplane which can be in one of three states : take-off, cruise, landing. The cruise state may be further de-composed as shown in figure 6.19.

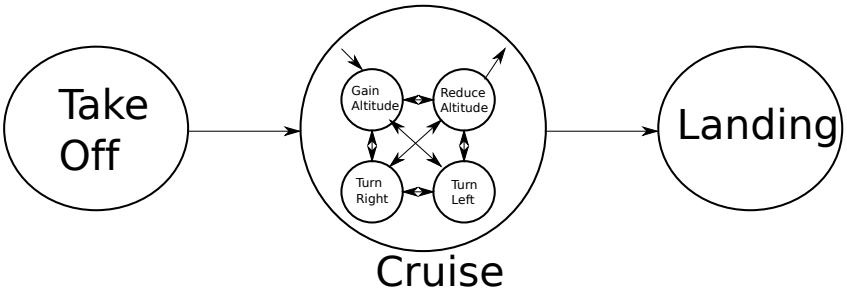


Figure 6.19: Example State Sequence for an Airplane

This type of Markov state model could be represented in RIIF using the notion of a timeline, as shown in figure 6.20. Once the time-line is defined, a set of *environments* can be defined. During each operational state, the environment in that state would determine operating *parameters* such as the workload, radiation flux and temperature.

This brief example is presented to highlight the need to model mission-profiles and show that this could be done through extensions to RIIF. Detailed definition of this aspect of RIIF remains future work.

```
timeline FLIGHT;

   initial state : TAKEOFF;
   assign TAKEOFF'duration = express_time( 5, "minutes" );
   assign TAKEOFF'next = CRUISE;

   state : CRUISE;
   assign CRUISE'duration = distribution {
                                    50 : express_time( 1, "hours" ),
                                    50 : express_time( 3, "hours" ) };

   assign CRUISE'next = distribution {
                                    99 : LANDING,
                                     1 : ABORTED_LANDING };

   terminal state : LANDING;
   terminal state : ABORTED_LANDING;

endtimeline
```

Figure 6.20: Code for a Timeline

6.5 Conclusion

The RIIF language does not contribute new concepts to reliability theory nor to the theory of modeling failures in ICs. Instead, it is a framework for expressing the existing theory in a standard, machine-readable format. This remains a key contribution, because in the absence of such a format, it is difficult to develop EDA tools which can analyze the reliability of complex SoCs. Today, models are built by manually entering data into spreadsheets; a process that is error-prone and which is not scalable.

It is clear, that the RIIF language is in an early state of development. Working through the set of examples presented in this chapter has identified certain issues which have been addressed. For example, the need to model time and to distinguish between a component's life-time and the time since it was powered-up was identified in section 6.3.3.

The examples also illustrate how the use of OO techniques allows the complexity of lower level models to be hidden. Significant effort is required to correctly model the rate of cell level upsets at different voltage conditions. Similarly, evaluating the AVF of structures in a processor is a challenging task. Today, these tasks are often performed by different types of engineers and rarely are the full models combined. RIIF provides a frame-work enabling detailed models to be developed by experts. These detailed models can then be combined to create a compact model of an entire chip, without compromising the modeling accuracy.

Chapter 7

Conclusions

Contents

There exists an enormous body of scientific work devoted to the analysis and mitigation of soft error effects and the topic continues to attract the attention of many researchers. The physics behind soft error phenomena are well understood although the impact of new process technologies such as FINFETs and FDSOI are still being studied. Despite this research, assessing the effect of soft errors on a complex SoC remains a challenging task and the approaches taken in industry often involve large approximations. The selective replacement of critical flip-flops is common practice, however, in industry the selection is often made based on simple heuristics and may be highly sub-optimal. The techniques proposed in this thesis make a modest contribution to better addressing these problems. As with any research, this document provides a snapshot of the current state of the work, however, additional research is required to further develop the techniques that have been proposed.

7.1 Simulation Analysis and Selective Mitigation

The first contribution of this thesis (section 3.2.4) involves using a CRC as a signature for the state of a digital system. When performing fault injection simulations, this CRC can be used to identify when the system with the fault has returned to its golden state. The advantage of the CRC is that it is compact. As presented in chapter 3, the CRC was calculated over the *entire* system, including the state of the flip-flops and the memories. In this way, the CRC provided a strong check of the full system, however, the calculation of the CRC required every bit in the design to be accessed.

The compute cost of the CRC can be reduced by calculating separate CRCs for different portions of the system. As shown in figure 7.1, a separate CRC trace can be created for each module in the design hierarchy. When a SEU is injected in a flip-flop in a specific module, such as B, then after the fault injection, the first

check is to see whether the CRC of module B matches the golden value. Assuming B represents only a small fraction of the design, this is much less costly than computing a CRC over the full system. If the CRC over B does not match, then it is certain the fault is still present and the simulation must be continued. If the CRC over module B matches, then either the fault has been masked, or it has propagated beyond B. At this point, the CRCs of the other modules can be checked, starting with those that are logically adjacent to B. In order to ensure the fault has been completely over-written, the CRCs of all modules must be checked, but this will occur only once, after the fault has disappeared. It is exepected that this extension would significantly reduce the cost of the CRC calculation. Furthermore, it would make it possible to study the propagation of faults through the design hierarchy. Development is underway to implement this enhancement and to quantify the benefit in terms of reduced simulation time.

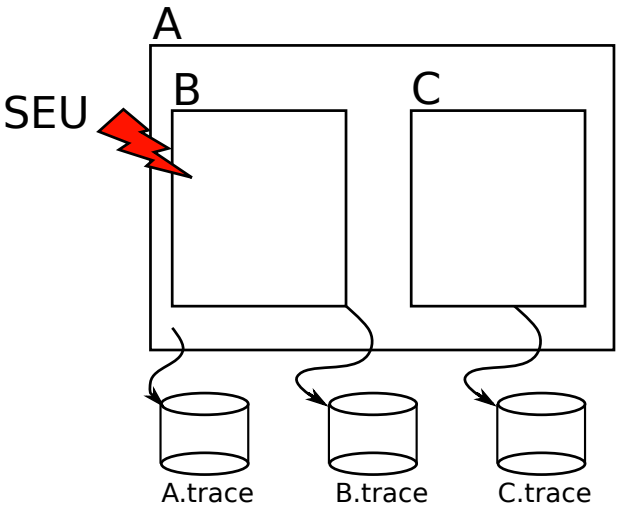


Figure 7.1: Calculation of Separate CRCs per Design Hierarchy

In chapter 4, techniques to cluster flip-flops were presented. By analyzing clusters, rather than individual flip-flops, far fewer fault-injections are required in order to identify which flip-flops to harden. Three different techniques for clustering were identified and compared. The technique that worked best consisted of heuristics that considered signal naming and module hierarchy. Since the original work in chapter 4 was performed, the proposed clustering techniques have been applied to a full networking ASIC. Through this work, several observations have come to light. The first is that the sizes of the clusters can vary significantly with some large clusters containing over ten thousand flip-flops. When these flip-flops are homogenous and have the same function, this is highly beneficial. In some rare cases, based on insights provided by designers, large clsuters were found to be heterogeneous and had to be manually split. This typically occurred when not all the bits within a

bus had the same function (some bits are used as control while others hold data). It would be beneficial if the algorithms could be extended to analyze whether the effect of the faults injected within a cluster are homogenous. In the event that they are not, this information could be used to propose an improved clustering.

Despite these challenges, clustering techniques have made it possible to effectively identify flip-flops for hardening on a large networking ASIC targeted to an application where an extremely low SDC rate is required. These techniques have provided a speedup of about two orders of magnitude and without them, it would be virtually impossible to use fault-injection simulations to identify critical flip-flops in a design of this scale.

7.2 Hierarchical SET Analysis

In chapter 5, a high-level methodology to quickly compute the effect of SETs in large ASICs was propoed. Following the initial work [Evans 2013], a simplified version of the methodology was implemented as a spreadsheet tool which estimates the contribution of SETs based on high-level design information (e.g. gate-count, distribution of gate-types, distribution of types of circuits). Unfortunately, the accuracy of such estimations is not well understood. When complex devices are tested under radiation, while executing real workloads, it is very difficult to distinguish between faults induced by SEUs and SETs. Most of the radiation test data for SETs is from simple structures such as chains of inverters or very basic circuits such as 4-bit adders. Extrapolating this data to a full chip and validating the result remains challenging.

To make progress, more test data is required on the actual contribution of SETs in circuits of moderate complexity. As part of a research project with the European Space Agency and the University of Saskatchewan, a test-chip is being developed which will include several medium sized combinatorial circuits (e.g. state-machines, arithmetic circuits, etc.). The combinatorial structures on the chip will be tested under radiation including under laser and heavy-ion micro beam. The measured data from the test-chip will be correlated both with aggregate and detailed (e.g. per-gate) SET estimations and in this way, the uncertainties will be quantified. A part of this work will also include extensions to better model the effect of EDR which was not considered in the methodology presented in chapter 5.

Although SETs have not traditionally been a concern in terrestrial applications, there is now a clear need to accurately estimate their contribution when performing chip-level SER analysis for high-end applications.

7.3 RIIF

Of the contributions of this thesis, the RIIF language is the one that has the potential to have the most impact. At the RIIF workshop held at DATE 2013, as well as at subsequent presentations, there has been genuine interest from industry in having a standard file format for specifying and exchanging the reliability data used in the

chip-design process. Based on the initial work presented in chapter 6, it appears that RIIF can be used to model radiation induced faults. However, there exist many other failure mechanisms for silicon devices [JEDEC 2010] and further work is required to investigate how RIIF can be extended to incorporate these. Some failure mechanisms such as Electro-Migration (EM) result in an explicit failure event, similar to radiation induced failures. Other mechanisms such as NBTI and HCI result in a shifting of device parameters. This shift may, or may not, provoke a failure. It remains to be seen whether this progressive degradation can be modeled in RIIF or whether it is better modeled through existing approaches such as aged SPICE models. Work is underway, in the context of the Modelling Reliability under Variability (MoRV) European FP7 project, to investigate new techniques to model these failure mechanisms and how to integrate these models into a design flow.

Beyond the question of the scope of RIIF, is the critical question of standardization. Even if the proposed language is *able* to model the occurence and the propagation of faults, it is of little value if there is no uptake of the language. There is thus a need to develop a formal specification of the RIIF language and then to work with multiple partners from industry to review the specification and move towards standardization. One of the challenges in proposing a new file-format is that it is only attractive when there is a critical mass of tools that can process files in that format. Therefore, another key component of the work ahead consists in improving the proto-type tool and, in parallel, working with other EDA companies to provide support for RIIF.

7.4 Summary

Without a doubt, in the years to come, electronics will play an increasingly critical role in all aspects of modern life. This is especially true as autonomous vehicules become a reality and as implantable medical devices incorporate new functonality. In these applications, the highest level of reliability is essential and there is a great need for techniques to evaluate complex systems and evaluate the impact of technology level faults. Analysis of complex systems requires *abstraction* in order to hide unnecessary details and *hierarchical decomposition* in order to scale. Both are widely employed in the silicon design process, whether it be design, functional verification, layout or timing analysis. To some extent, reliability analysis has been slow to adopt these approaches. The key contribution of this thesis is to show how *abstraction* and *hierarchical decomposition* can be applied to better analyze and mitigate the effect of faults in large, complex designs. Subsequent to the initial work, many of the techniques have been applied to production designs.

Appendix A

Use of Bloom Filters to Protect TCAMs

This appendix presents the results of a collaborative research project between University Rome 'Tor Vergata', TIMA Laboratory and Cisco Systems. The content is based on a paper which was presented at DATE 2013 [Pontarelli 2013b].

A.1 Introduction

A Content Addressable Memory (CAM) memory is capable of comparing a word of input data against all the data words stored in the memory, and provides as output the address of the first stored word which matches the input data. In a binary CAM, the memory entries are composed only of 0's and 1's. A TCAM also allows the entries stored in the memory to contain 'X' bits which represent a *don't care* value. This provides a compact means to store a large set of match patterns. For example an entry *10XX1* in a TCAM matches the following four patterns *10001*, *10011*, *10101*, *10111*. The typical implementation of the *don't care* bits is obtained by defining a mask string for each entry of the TCAM. In the previous example the mask string would be *00110* while the actual entry is *10--1* where - can be either 0 or 1. TCAMs are widely used in high speed network systems to implement features such as classification and access control [Chao 2002]. In fact, their ability to perform massive comparisons with O(1) complexity makes them extremely appealing for searching large tables with very low latency. The use of the *X's* makes it possible to apply admission or quality of service (QoS) policies to classes of traffic using a modest number of table entries.

Currently, the use of nanometer scale technology, the reduction in operating voltages and the increase in the overall number of stored bits have caused a consequent increase in the rate of occurrence of SEUs. ECC is highly effective at protecting regular memories. It is difficult to apply techniques based on information redundancy to TCAMs because all entries are accessed simultaneously, thus the actual checking circuitry must also be replicated for each entry. Networking systems based on TCAMs require high reliability and new approaches are required to mitigate the effects of SEUs.

In the literature different techniques have been proposed to mitigate the effects of SEUs in TCAMs. Most techniques operate at the circuit level and require modifications to the CAM architecture [Krishnan 2009] or are based on a background parity scan which has a long detection latency.

The proposed technique does not require modifications to the internal structure of the TCAM since this is often a discrete component designed in a full custom design flow to minimize power consumption and optimize the density of memory cells. The proposed technique can be implemented as a stand-alone module that operates in parallel with the TCAM and checks the correctness of the results of each access. Error detection is immediate, unlike techniques based on background parity scans. Moreover, the proposed solution is particularly well suited to TCAM applications with *wide* word sizes, like the ones used in modern network systems, since the overhead is independent on the TCAM word size.

In previous work [Pontarelli 2013a], a technique to add a Bloom Filter in parallel to a *binary* CAM was described. The technique described in this chapter extends the technique to TCAMs.

A.2 Existing TCAM Protection Techniques

In industry, the standard technique to protect TCAMs against radiation induced errors is to protect each word with a parity code. It is not practical to simultaneously check the parity of all entries and it is not sufficient to only check the parity of the matching entry, since an error on a higher priority entry can create an incorrect match. Therefore, a parity scan engine runs in the background and sequentially reads through each of the entries, checking the parity and triggering an error indication if there is a mismatch. When an error is found, software intervenes and re-writes the corrupted TCAM entry. The drawback to this method is that there can be a significant latency between when an error occurs and when it is detected and then fixed by software. During this window of time, incorrect decisions are occurring. Multiple interleaved parity bits are used to protect against upset events which corrupt adjacent cells which is an increasing concern.

In [Krishnan 2009], error correcting codes for TCAMs are proposed, however, these require a 200% overhead in order to correct single bit errors. In [Azizi 2006], a hardened TCAM cell with cross coupled feedback loops is proposed. A SER reduction of about 30% is achieved with an area penalty of 15%. US Patent 7254,748 describes a technique to duplicate TCAM cells to provide error detection and correction, however, this represents and area and power overhead of 200%. None of the existing solutions provide strong, on-line error detection with a modest area overhead.

A.3 Background on Bloom Filters

A Bloom Filter [Bloom 1970] is a probabilistic data structure based on hashes and used to check the membership of an element in a set. A Bloom Filter performs two tasks: 1) stores a set of items in its memory, and 2) quickly responds to a query about the presence of an item. The drawback of using this structure lies in its probabilistic nature which can return false positives. With a certain probability

(called the false positive rate), the Bloom Filter can report a data element as being present when in fact it was not inserted into the Bloom Filter. A false positive occurs when the hash functions alias. However, when a Bloom Filter signals that a data is not present, this is always the case (i.e. a false negative never occurs in a Bloom Filter).

A Bloom Filter is implemented as a bit array of m bits accessed via k hash functions $H_1(x)...H_k(x)$. Each hash function maps a set member x to one of the m bits within the bit array. We denote as $v(i)$ the value of bit i within the bit array. Figure A.1 illustrates the structure of a Bloom Filter.

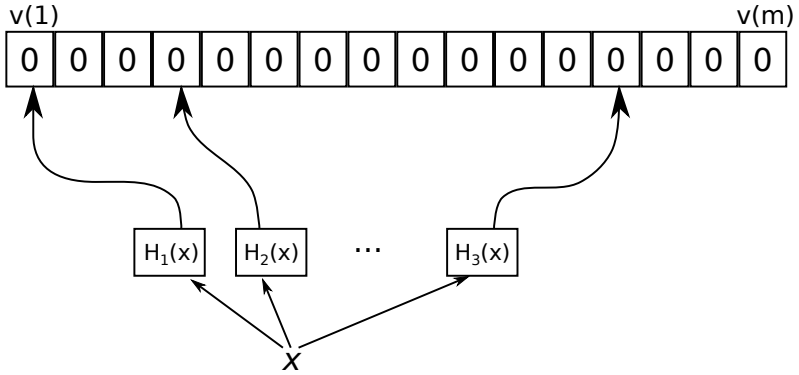


Figure A.1: Structure of a Bloom Filter

Two operations are possible with a Bloom Filter:

1. **Insertion:** An element x is inserted into the filter by setting to one all the indexes of the bit array addressed by the k hash functions. In a mathematical notation this corresponds to:
   $\forall i \in \{1..k\}$, $v(H_i(x)) \leftarrow 1$

2. **Querying :** An element is reported as present in the filter if all the values of the bit array addressed by the k hash functions are equal to 1.
   $result \leftarrow min\{v(H_i(x))\}, i \in \{1..k\}$

In other words, when a word x is inserted, the bits corresponding to the result of hashing x with all the hash functions are set to 1. An element is then said to be in the set if all the bit positions to which it is hashed are set to 1. Clearly, there can be no false negatives. It is, however, possible that the bits set by two or more words actually stored in the array correspond to the bits of another word x which is being queried but which is not stored in the set, resulting in a false positive.

For a Bloom Filter in which n elements are stored, the probability $p(n)$ that a given bit in the filter is zero is given by:

$$p(n) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-n\frac{k}{m}} \tag{A.1}$$

If we test membership of an element that is not in the set, each of the k bit array values indexed by the hash is 1 with probability $1 - p(n)$. The probability of all of them being 1, which would cause the false positive, is then:

$$P_{fp}(n) = (1 - p(n))^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \tag{A.2}$$

The probability of false positives decreases as m, the number of bits in the array, increases. The probability of false positives increases as n, the number of words stored in the filter, increases. For a given m and n, the value of k (the number of hash functions) that minimizes the probability of false positives is: $k = \frac{m}{n} \cdot ln2 \approx 0.7 \cdot \frac{m}{n}$.

Using equation A.2, we can size the Bloom Filter according to a required number of elements to be stored and a required minimum false positive rate.

A.4 Application Aware Hashing

A trivial method of inserting *don't care* bits in a Bloom Filter is to fully expand each of the *don't care* bits. This is not scalable as the number of insert operations and the size of the Bloom filter grows exponentially with the number of *don't care* bits and essentially this consists of treating the TCAM as a binary CAM.

From a purely hardware perspective a TCAM can have *don't cares* in any bit of any entry. However, some practical considerations can be used to identify patterns within the entries based on how TCAMs are used in industrial applications. In networking systems, a single large TCAM is used to support multiple features or applications. Examples of such applications are packet classification, Access Control Lists (ACLs), traffic sampling (Netflow) and QoS - to cite the most common. The entries in the TCAM are divided based on the applications and a small number of bits within the lookup word are used to select which set of entries to match when a given lookup is performed for a specific application. For a given feature, a specific set of fields from the packet header are looked up. The usage of fields within an application is regular and there are patterns in where *don't care* bits appear for each application. This regularity of the *don't care* bits can be exploited to avoid the explosion that would result from a naïve expansion of *don't care* bits.

TCAM entries can be divided into fields of which we identify three types:

1. **Fully Unmasked** This type of field corresponds to groups of N bits which must match exactly. These entries correspond to a binary CAM.

2. **Prefix** This represent any field of size N bits where the most significant M bits are unmasked (must match) and the least significant $(N - M)$ bits are masked (*don't care*).

3. **Generic** This represents an arbitrary field of N bits where any combination of bits within the field may be masked or unmasked and the combination of

don't care bits can appear anywhere.

TCAM configurations are usually computed by algorithms that transform classification rules based on port ranges, IP addresses and other packet header fields into a set of TCAM entries [Lakshminarayanan 2005]. For each rule, the algorithm generates a set of similar entries. The packet fields that are relevant for classification vary from one feature to the next (e.g. QoS, ACLs, etc.) thus different similarities occur depending on the feature. For this reason, an *application aware* approach to building the Bloom Filter is proposed. The key idea to avoid state explosion is to select a different set of hash functions for each feature in order minimize the *don't care* bits. This approach is called *application aware hashing*.

In particular, for fields that operate on a *longest prefix match*[1], we use the approach proposed in [Dharmapurikar 2006]. This method partially masks the least significant bits, hashing only a variable prefix part of the field. The hash functions are grouped with respect to an interval of the prefix length. During the insertion phase, the field is masked following the given prefix *don't care* configuration, and is inserted in the Bloom Filter using the whole set of hash functions. During the search phase, the item to be searched, is progressively masked and queried in the Bloom Filter, as depicted in figure A.2). The example in the figure illustrates an old, classful networking hierarchy, however, the approach in [Dharmapurikar 2006] also works for arbitrary prefix lengths.

Finally, for generic fields, we propose a hashing methodology aimed at compacting the number of *don't care* bits. The regularity of TCAM configurations allows us to define some rules, that can be applied to perform the *application aware hashing*. The most significant bits of the TCAM entry identify the feature and define the format of the subsequent fields of the entry. The following rules are applied:

1. Unmasked fields can be used directly as input of the hash functions.

2. Fields that are always masked can be ignored in all the hash functions.

3. Fields of prefix type can be managed using the *longest prefix match* algorithm.

4. Otherwise masked bits are compressed in order to reduce the number of *don't care* bits (e.g. XORing contiguous bits).

[1] Routing tables consist of an address and a subnet mask. When a packet address is looked up, the routing table entry with the longest match, based on the subnet mask, is selected. Normally, TCAMs are not used to store the main routing tables, as more efficient data structures exist, however, sometimes smaller address tables are stored in TCAMs. In this case, they contain entries sorted by the length of the subnet mask.

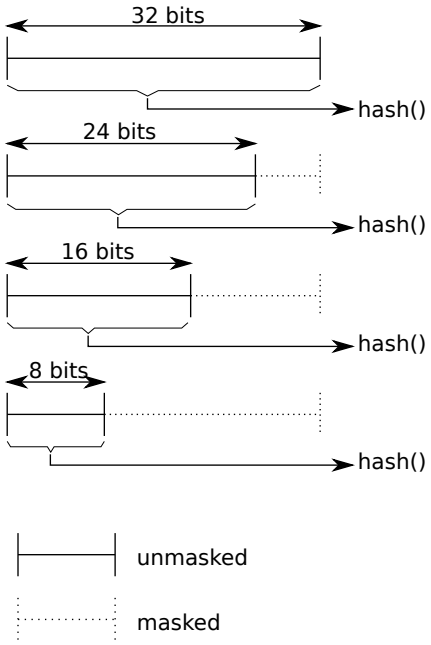


Figure A.2: Masking of Bits for Longest Prefix Match

In order to illustrate the method, a sample 16-bit wide TCAM configuration is shown in table A.1. In this example, the TCAM contains four features:

- Feature 1: In this feature, the TCAM is configured to act as a binary CAM. The entries for this feature are composed of three fields. The first field (00) identifies the feature. The second field represents bits that are not used and is always masked and the third field is always unmasked. This feature acts as a 10 bit binary CAM. Based on the above rules, only bits $in[9..0]$ of the input data is hashed.

- Feature 2: The feature is composed of four fields. The first two bits (01) identify the feature, the second field, composed of two bits, is always unmasked. The third field is generic (i.e. can be masked or unmasked), while the last field is not used and is always unmasked. For this feature, the input to the hash function is the concatenation of three bit vectors: $\{in[13..12], (in[11] \bigoplus in[10] \bigoplus in[9] \bigoplus in[8]), in[7..0]$ Using this vector as input for the Bloom Filter, when the third field presents unmasked entries, a regular item is inserted in the Bloom Filter. For the masked entries (e.g. the entries with the X's) the XOR compression reduces the number of X's in the input to the Bloom Filter to a single bit. At this stage we can expand the *don't care*, inserting the two items corresponding to '0' and '1' into the Bloom Filter. This type of compression works well for generic fields where the values of this fields often present some regularities that are repeated among the different entries.

- Feature 3: This feature is similar to feature 1 using this part of the TCAM as a binary CAM. The first field (10), identifies this feature, the second field is unmasked and the third field is always masked. In this case, the input for the Bloom Filter are the bits $in[13...8]$ of the original input data.

- Feature 4: This feature is composed of three fields. The first field identifies the feature (11) and the second is a 6 bits unmasked field which functions as a binary CAM. The third field uses *longest prefix match*. Applying the rules for *application aware hashing*, the full input vector is given to the *longest prefix matching* algorithm, iteratively setting the range for bit masking for the bits from 0 to 7.

Table A.1: Example TCAM Configuration

TCAM				
Index	Feature	Key		
Feature 1 - Binary CAM				
		Always Masked		Unmasked
0	00	XXXX		01_0000_0001
1	00	XXXX		10_0010_0001
2	00	XXXX		01_0001_0111
Feature 2 - Generic Patterns				
		Unmasked	Generic	Unmasked
3	01	00	0001	1110_1111
4	01	01	0010	1010_1110
5	01	01	0000	1010_0011
6	01	11	0101	1010_0001
7	01	11	XXXX	0011_1001
8	01	00	XXXX	1100_1100
9	01	00	XXXX	0000_0110
10	01	00	XXXX	0001_0011
Feature 3 - Binary CAM				
		Unmasked		Always Masked
11	10	00_0000		XXXX_XXXX
12	10	00_0011		XXXX_XXXX
Feature 4 - Longest Prefix				
		Unmasked		Prefix
13	11	10_0010		01XX_XXXX
14	11	01_0000		1101_XXXX
15	11	11_0000		1100_00XX

A.5 Error Detection Architecture

In this section, two architectures for detecting errors in TCAMs are described. Both architectures use *application aware hashing* to insert and query items in a set of Bloom Filters. The first architecture (figure A.3) uses two Bloom Filters, one that take as input the same key that is applied to the TCAM, while the other takes as input the concatenation of the tuple $key, entry$, where $entry$ is the response of the TCAM to the key input. The first Bloom Filter detects the occurrence of a SEU causing a pseudo-MISS [Pontarelli 2010], i.e. the TCAM reports that the searched key is not present. If this type of error occurs, the output of the first Bloom Filter and that of the TCAM are discordant, the TCAM gives a MISS, while the Bloom Filter gives a HIT. The second Bloom Filter detects the occurrence of pseudo-HIT or multi-HIT errors [Pontarelli 2010], i.e. the TCAM returns an erroneous entry or a MISS condition. This error is detected by the second Bloom Filter, which signals that the tuple $\{key, entry\}$ in not present in the TCAM. If, due to hash collision, a false positive occurs in the Bloom Filter the error is undetected. The experimental data presented later shows that the occurrence of both an error in the TCAM and a false positive in the Bloom Filter are very low, even with a small Bloom Filter. The experimental data shows up to 100% error detection, when the Bloom Filters are large enough to avoid most false positives. The two Bloom Filters detect disjoint types of errors and the fraction of errors detected by the two Bloom Filters strictly depends on the TCAM configuration and on the input keys applied to the TCAM. In particular, when the TCAM is configured to be used as a full classifier [Lakshminarayanan 2005], the first Bloom Filter becomes useless, since there are never any misses in the error-free TCAM.

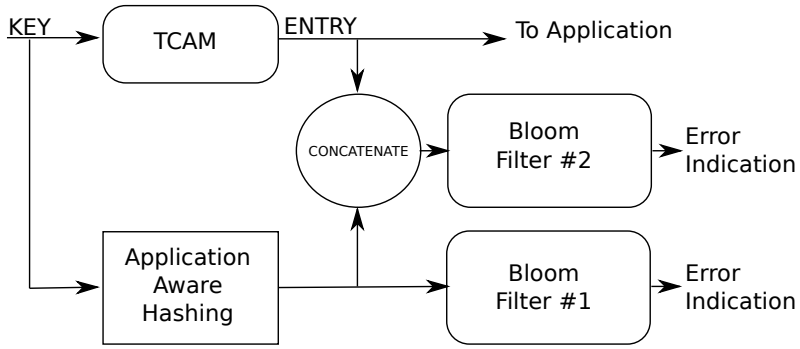


Figure A.3: Architecture Using Two Bloom Filters

The second architecture shown in figure A.4 uses M Bloom Filters and splits the content stored in the TCAM into M different subsets. Each item is inserted in only the i^{th} Bloom Filter, corresponding to the condition $i = entry mod M$. Since the false positive rate of Bloom Filters depends on the ratio between the number of inserted items and the size of the Bloom Filter, the false positive rate is independent of M, the number of sets into which the TCAM configuration set is divided. Unlike the

first architecture, this architecture presents a lower error detection latency, because the query to the M Bloom Filters can all be done in parallel with the TCAM lookup. The TCAM output is only used to select which of the M responses from the Bloom Filters should be considered. The drawback of this architecture is that undetected errors can be caused both by hash collision, like in the previous architecture, or when an erroneous entry (i.e. the response of a TCAM affected by a SEU) and the correct one (i.e. the one stored in the Bloom Filter set) have the same value modulo M. The use of M = 16 or M = 32 allows us to reduce this aliasing probability to 6% or 3%. When no false positive occurs, the set of Bloom Filters provides a one-hot encoded word as output, while with the occurrence of a false positive in some of the Bloom Filters that do not correspond to the requested key, the output word has multiple bits to 1. If a false positive occurs simultaneously with an error in the TCAM, the error is undetected if the i^{th} index given by the modular operation selects one of the Bloom Filters affected by the false positive.

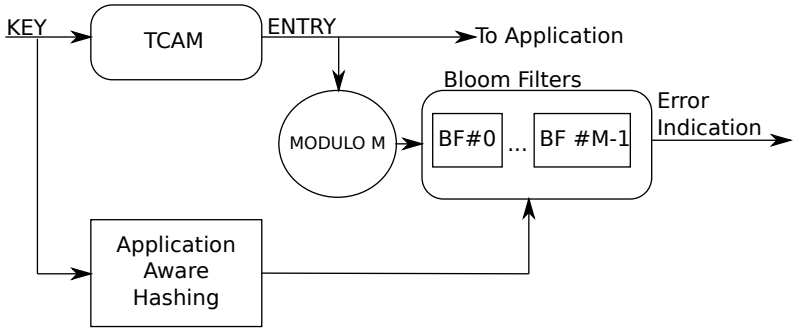


Figure A.4: Modular Detection Architecture

The algorithm for error detection using the second architecture is shown in figure A.5. First the algorithm checks if all the Bloom Filters are zero. This corresponds to the condition of a MISS in the TCAM. If the TCAM gives MISS as a response, then no error has occurred. Otherwise, if the TCAM responds with an entry we assume that a pseudo HIT error has occurred. Instead, if some of the Bloom Filters give 1 as a response, we select the Bloom Filter with index i, where i is the entry number returned by the TCAM modulo M. If the i^{th} Bloom Filter returned 1, we assume that no error has occurred, otherwise an error in the TCAM is reported. The condition of BF(i) = 1 can be caused by the simultaneous occurrence of a false positive in the Bloom Filter and by an error in the TCAM. The experimental results show that this is very unlikely.

A.6 Experimental Results

In order to evaluate the effectiveness of the proposed methods, a set of simulations have been performed. A C++ software model of TCAM and Bloom Filters was designed and both the TCAM and the Bloom Filters were configured using a real

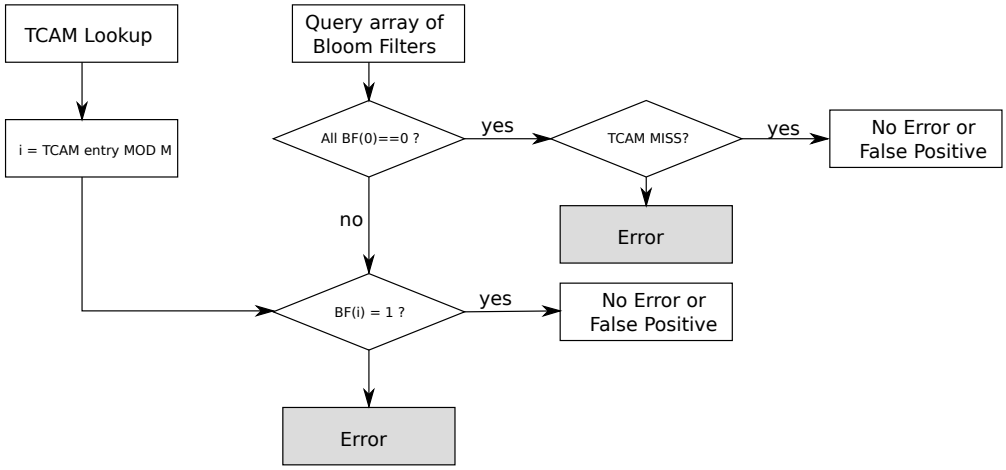


Figure A.5: Detection Algorithm with Modular Bloom Filters

data set provided by Cisco Systems. Ten thousand simulations were run where a fault was injected into the TCAM for each run. The activation of the fault was determined by comparing the output of the TCAM under test, with the output from a golden TCAM. For the activated faults, the number of faults detected by the proposed architectures is reported. We compute the area overhead as the number of bits of storage in the Bloom Filter with respect to the number of configuration bits of the TCAM. This is a somewhat pessimistic, as the real silicon area required to implement a glsTCAM cell is roughly twice the area of an SRAM cell, since the glsTCAM cell also contains matching logic. We do not evaluate the logic required to implement the hash function of the Bloom Filters, since it is composed of only a modest number of XOR gates and is small compared to the area required for the memory. The two architectures presented in the previous section have been simulated, and the size of the Bloom Filters has been varied, in order to study the impact of false positives on the efficiency of the proposed methods. For the second architecture the analysis is limited to M = 4, since the TCAM configuration was too small to benefit from a larger value of M.

The results reported in table A.2 show that the errors in the TCAM are detected by one of the two Bloom Filters used in the first architecture (see figure A.3). As expected, the set of errors detected by the two Filters is disjoint, and the two Filters are able to detect 100% of injected faults when the size of the Bloom Filters is sufficiently large. Even without considering that SRAM memory cells are smaller than TCAM cells, the ratio 8% is less than the nominal 12.5% overhead that would be required for byte-wise parity. When the overhead is further reduced to 4% or 2% the effect of false positives in the Bloom Filters produces some undetected errors, decreasing the error coverage to 98.7% and 89.1% respectively.

In table A.3, the results for the second architecture (see figure A.4) are reported. As expected, even with a large Bloom Filter, the error coverage is less than 100%, due to the aliasing of the modular operation. However, the ratio of undetected

Table A.2: Simulated Results Using Two Bloom Filter Architecture

Size of Single Bloom Filter (bits)	Errors Detected by Filter #1	Errors Detected by Filter #2	Total Error Coverage	Total Overhead
2K	2013	1307	100%	32%
1K	2332	1306	100%	16%
512	2343	1242	100%	8%
256	2068	1342	98.7%	4%
128	1911	1168	89.1%	2%

errors is less than expected, (should be 25%, since M = 4), since the aliasing does not occur when a fault inside the TCAM produces a pseudo-HIT. In fact, in this case, all the Bloom Filters give 0 as response, while the TCAM signals the presence of the key in an erroneous entry. The condition that all Bloom Filters are equal to 0 allows the error to be detected without any modular operation, and therefore without the issue of aliasing. Aliasing still remains an issue for multi-HIT errors. When the size of the filters is sufficiently large, no false positives occur, and the error coverage is independent of the size of the Bloom Filters.

Table A.3: Simulated Results Using Modular Bloom Filter Architecture

Size of Single Bloom Filter (bits)	Number of Errors Detected	Total Number of Errors	Total Error Coverage	Total Overhead
2K	3211	3662	87.6%	32%
1K	2908	3326	87.4%	16%
512	3010	3461	86.9%	8%
256	3130	3669	85.3%	4%
128	4276	3507	82%	2%

A.7 Conclusions

The result of the work presented in this chapter is a technique that can effectively detect errors in TCAMs. The method is attractive because it can be implemented as a circuit that operates in parallel with an existing TCAM device, thus it can be applied to discrete TCAM memories. With the standard approach to TCAM error detection, based on a background scrub process, there is a long error detection latency, however, the proposed approach detects errors when they occur. Furthermore, the technique works even if many bits in the TCAM memory are upset and is not limited to detecting single bit errors.

The core of the proposed technique is an *application aware hashing* algorithm

that, for practical configuration patterns, enables the *don't care* bits in the entries to be effectively stored in the Bloom Filter. This technique is not only attractive for error detection, but it also has applications for reducing power consumption. Commercial TCAM devices are physically divided into banks and when a search request is issued, in addition to the search key, a bank mask is provided, indicating which banks should be enabled for that search. Of course, the fewer banks that are enabled, the less energy that is consumed. A set of Bloom Filters can be placed on-chip, as shown in figure A.6. Each on-chip Bloom Filter stores the contents of the corresponding bank in the off-chip TCAM. Before a search key is issued externally, the on-chip Bloom Filters can be queried. For those filters which indicate that the key is not present in the corresponding bank, the external bank need not be searched. In this way, only a small number of banks need be enabled. There are no false negatives, so the correct bank is always searched. Of course, false positives can occur, but the only drawback is that the corresponding bank in the external TCAM is searched unnecessarily. Further research is underway to quantify the power saving benefits and to potentially implement this technique on a commercial ASIC.

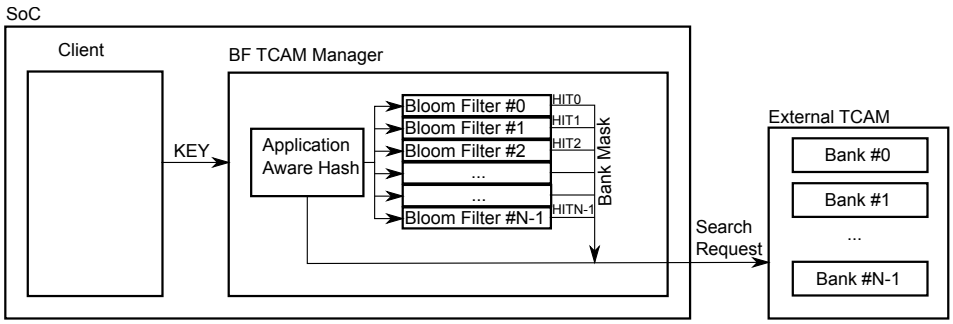


Figure A.6: Use of Bloom Filters to Reduce TCAM Power Consumption

Appendix B

Approximate Logic Functions for Protection of Combinatorial Logic

This appendix presents the results of a collaborative research project between the University of Saskatchewan (Canada), iROC Technologies and Cisco Systems. The content is based on a paper which was presented at SELSE 2014 [Xie 2014].

B.1 Introduction to Logic Repair

In chapter 1, it was identified that faults in combinatorial logic are more difficult to analyze and mitigate than those in memories or sequential elements. In modern technologies, permanent faults are a growing concern and it is now standard to implement in-field repair techniques for memories. Equivalent repair techniques do not exist for combinatorial logic, although different approaches have been proposed. Redundant logic is a technique that can mask both transient and permanent faults in combinatorial logic. Generally, the approaches to manage faults in combinatorial logic can be divided into three categories : approaches based on coding, approaches using on-line test and approaches based on defect aware synthesis.

Many authors have studied extending the coding techniques used for memories (Hamming, Berger, weight-based codes) to *detect* errors in combinatorial logic [Das 2000, Leveugle 2002]. In [Dutta 2008], a selective error *correction* technique is proposed with area overheads from 49% to 153%, however, it results in many additional layers of logic and is thus not applicable to high-speed designs.

Another approach to manage faults in combinatorial logic is through on-line self test and selective hardware sparing. On-line testing is possible when portions of a circuit can be periodically taken off-line for test. For example, in [Li 2008], N+1 sparing is implemented on a 16 channel DMA controller to achieve an 87% repair coverage with a small (6.3%) area overhead. Approaches based on on-line testing can be highly effective; however, they remain application specific. Special techniques exist for specific types of circuits such as arithmetic circuits [Nicolaidis 1997, Nicolaidis 1998] and digital filters [Bayraktaroglu 2000] but there do not exist general techniques to add on-line testing and sparing to a generic digital circuit. Normally, the goal of logic synthesis algorithms is to produce a circuit with minimum area within a given timing constraint, thus redundant logic is eliminated. In fault-tolerant synthesis, the mapped circuit intentionally contains redundancy in order to mitigate faults. Redundant logic functions [Mohanram 2003, Sierawski 2006, Sanchez-Clemente 2012, Adeleke 2012, Yuan 2013, Choudhury 2013] fall into this

category. Similar work in this area also includes [Zheng 2009], where the authors propose a technique for mapping logic to PLAs that have known defects. In [Almukhaizim 2008], redundant wires are inserted in a combinatorial circuit, specifically to reduce the sensitivity to transients.

In early work on approximate logic functions [Sierawski 2006], a BDD representation of the circuit was built and then pruned to produce approximate functions. In [Sanchez-Clemente 2012], the authors perform a unate decomposition of the original function and then simplify this by eliminating those lines with a testability metric below a given threshold. The coverage of the resulting circuit is evaluated with stuck-at and fixed-width transient faults. Other authors [Choudhury 2008, Choudhury 2013] have used a Sum of Products (SOP) representation to build the approximate logic function, which is the same representation used in this work. In [Choudhury 2008], the authors use approximate logic for Concurrent Error Detection (CED) and extend this work in [Choudhury 2013] to the masking of stuck-at and timing faults. Their algorithm does not consider the actual per-gate failure rates. The primary contribution of the algorithm proposed in this chapter is the fact that it considers the per-gate failure rates. This is achieved by incrementally performing fault injection simulations as the sequence of approximate functions is generated. Through careful consideration of the statistical confidence intervals, the number of fault injections is kept to the minimum number required to identify the next best cube to select. For several circuits, we illustrate the trade-off between masking and increased area/power and for some circuits, we show higher quality results than those obtained in previous work.

B.2 Overview of Approximate Logic Functions

The basic idea behind redundant logic is that any combinatorial logic function, G, can be augmented with a simpler approximation of the min-terms (F) and the max-terms (H), as in figure B.1. For any input vector for which F is true, G is also true. In this way, if there is a fault causing G to incorrectly be false, then it may be corrected by F. In a similar way, H may fix faults where G is incorrectly true. Potentially, F or H can be empty (e.g. H=1, F=0). The maximum added delay is two gates, and if either F or H is empty, the delay is a single gate. The challenge is to generate F and H such that they mask a maximum number of faults while minimizing the area and power overhead.

The failure rates of gates can vary significantly. Clearly, an inverter consisting of two transistors is less prone to stuck-at faults than a complex gate with 10 transistors like an XOR, as seen in figure B.2. For this reason, it is important to consider the actual technology failure rates when synthesizing redundant logic functions. We assume that the per-gate FIT rate, f_i, has been characterized in advance.

Due to logical masking effects[1], a fault in a given gate has a probability of

[1]We primarily consider the effect of logical masking. However, since the approach is based on gate-level fault injections, the effect of electrical masking or pulse stretching would be taken into

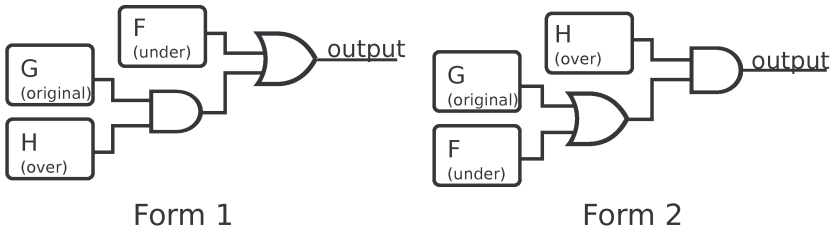


Figure B.1: Overview of Approximate Logic Functions

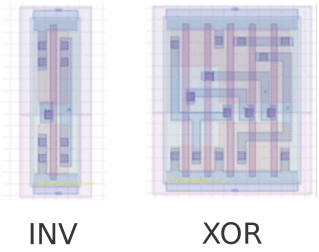


Figure B.2: Layout of Two Gates from Nangate Library

propagation to the outputs that we designate as $EPP_i(V)$, where V is a selected set of input vectors. The FIT rate of the full circuit is given by equation B.1.

$$FIT_{cct} = \sum_{i \in Gates} f_i \cdot EPP_i(V \tag{B.1}$$

The goal is to reduce the circuit level failure rate using redundant logic, while minimizing the area and power overhead. We evaluate FIT_{cct} , using equation B.1, both for the original circuit and then for the protected circuit with the redundant logic. Of course, when evaluating the protected circuit, the failure rate of the gates in the redundant logic is included, thus, to yield a *net* improvement in reliability, the fault masking of the redundant gates must more than offset the increased intrinsic failure rate due to a higher gate count.

B.3 Proposed Algorithm

Most problems in logic synthesis are NP-hard [de Micheli 1994] thus heuristics must be used to find solutions in reasonable time. The starting point for the proposed algorithm is a minimal two-level representation of the original logic function, expressed as a list of cubes, as produced by the Espresso logic minimizer [Rudell 1987]. Consider an example logic function with two outputs, G0 and G1, whose Karnaugh map is shown in figure B.3. After logic minimization, the cubes are shown in table B.1.

The proposed algorithm incrementally selects cubes from this list to add to the F or H approximate functions as shown in figure B.4.

account, if the technique were extended to SETs.

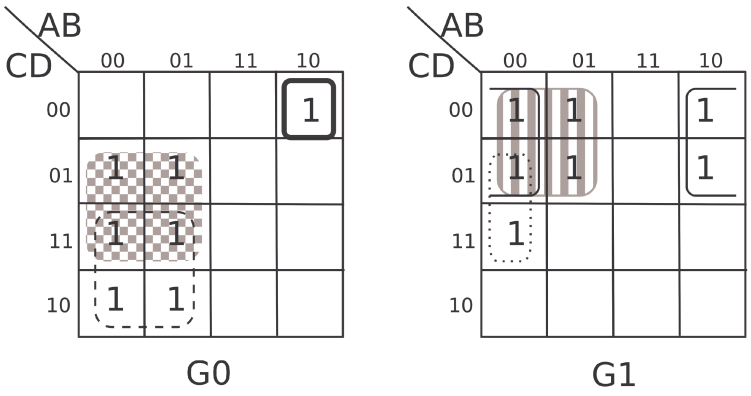


Figure B.3: Karnaugh Map of Example Function

Table B.1: Logic Cubes for Example Circuit

G0		
`0--1`	max	4
`0-1-`	max	4
`1000`	max	1
`11--`	min	4
`1--1`	min	4
`1-1-`	min	4
`0-00`	min	2
G1		
`0-0-`	max	4
`-00-`	max	4
`00-1`	max	2
`-11-`	min	4
`--10`	min	4
`11--`	min	4
`1-1-`	min	4

A Fault Reduction (FR) metric is computed for each cube using equation B.2. The coverage of the cube is trivial to obtain from the two-level representation and measures the number of input vectors that will be protected by the cube. For example, the number of vectors covered by `0--1` is 4 (`0001`, `0011`, `0101` and `0111`). The EPP is the likelihood that errors in the gates actually affect one of the outputs, for the vectors covered by the cube. This probability is estimated through fault-injection simulations.

$$FR(cube) = coverage(cube) \cdot \widehat{EPP_i} \tag{B.2}$$

The cubes are ranked and the best cube is selected for addition to the approx-

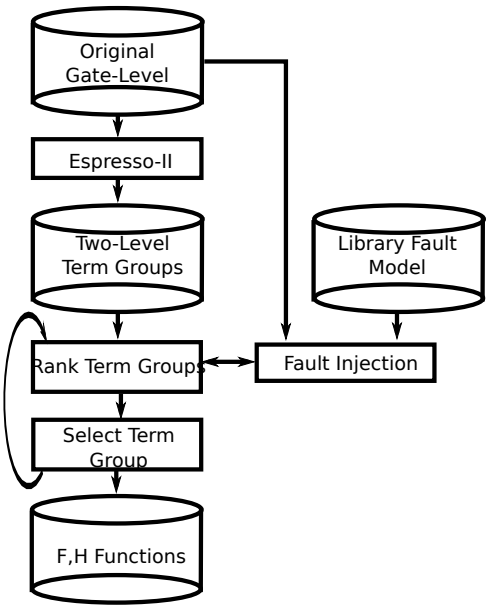


Figure B.4: Algorithm for Generating Redundant Logic

imate function (F or H). After selecting one cube, the FR metric for the remaining cubes is re-calculated, in a lazy fashion described below. The cubes are then re-sorted to identify the next best candidate.

The EPP is determined using statistical fault injection performed on the gate-level circuit using a digital simulator. Random input vectors, within the scope of the cube, are generated based on the distribution of vectors in V (a set of vectors taken from an input trace). For each fault-injection, one gate is randomly chosen, weighted by the per-gate FIT rates, f_i. A stuck at fault (0 or 1) is injected on the selected gate and it is determined if the fault propagates to a primary output. This biased fault injection approach was chosen in order to take into account the actual expected failure rates of the gates in the mapped circuit. This ensures that the redundant logic functions are optimized to cover the faults that are most likely to occur.

The objective is to perform the minimum number of fault injections necessary in order to identify the next best term to select. To avoid running more simulations than necessary, the simulations are launched in small batches of size N_{batch}. After the first batch of simulations, the value of EPP is estimated ($\widehat{epp}_n$), based on the fraction of simulations where the fault propagates to an output. If we assume that the distribution of $\widehat{epp}_n$ is normal, then the confidence interval is given by equation B.3 [Wasserman 2005, p. 130], as discussed in section 3.3.

$$\widehat{epp}_n \pm z_{\alpha/2}\sqrt{\frac{\widehat{epp}_n(1-\widehat{epp}_n)}{n}} \tag{B.3}$$

Using the number of cubes and the estimated EPP, the FR is computed for each cube and the cubes are ranked. Due to the error bars, at the top of the list, there may be multiple overlapping best candidates. In this case, additional fault-injections are performed, in groups of N_{step}, in order to further reduce the error bars. Additional simulations are only run for those cubes that are potentially the best candidate, based on their error bars. A maximum of N_{max} simulations are performed and if there are still multiple candidates, the term with the best FR is simply selected, disregarding the other candidates whose FR estimate may overlap. In these cases, there is little difference between the best candidates.

Consider the example logic function shown in figure B.3. After the first round of the algorithm, with 20 fault-injections, the results are shown in table B.2 . The term `0--1` has the highest FR metric, however, there are three other terms (`-11-`,`--10`,`11--`) with FR metrics that could overlap, due to the error bars. The remaining ten terms (shown in grey) can be eliminated as their FR metric is too low, even considering the error bars, which are calculated using equation B.3.

Table B.2: Results after First Round

Inputs	Output	Type	Num Terms	Num FI	FR	Error Bar
`0--1`	G0	Max	4	20	2.6	0.546
`-11-`	G1	Min	4	20	1.8	0.5696
`--10`	G1	Min	4	20	1.8	0.5696
`11--`	G0	Min	4	20	1.8	0.5696
`1--1`	G0	Min	4	20	1.4	0 546
`0-1-`	G0	Max	4	20	1.4	0 546
`11--`	G1	Min	4	20	1.2	0 5248
`1-1-`	G0	Min	4	20	1.2	0 5248
`1-1-`	G1	Min	4	20	1.0	0 4956
`0-00`	G0	Min	2	20	0.8	0 2804
`00-1`	G1	Max	2	20	0.7	0 273
`1000`	G0	Max	1	20	0.7	0 1312
`-00-`	G1	Max	4	20	0.6	0 4088
`0-0-`	G1	Max	4	20	0.4	0 3436

After additional fault injections, the term `0--1` is selected as the first term to add to F. Before proceeding to the next round, the table of min/max-terms must be updated to reflect the overlap between `0--1` and the other terms. For example, the number of terms for the cube `0-1-` is reduced to 2 (`0011` and `0111`).

For the next round, new fault injections must be performed to reflect the coverage provided by the redundant logic that has been added. Note, however, that this is only necessary for terms that overlap with the newly added, redundant term.

Furthermore, the new FR can only be lower than the FR metric from the previous round, thus the previous FR estimate serves as a valid upper bound. Additional fault-injections are only performed for those terms which are potentially the next

best-candidate in the following round. In this way, a sequence of F, H functions are generated with each one providing increasingly better coverage of G, taking into account the actual failure rates of the library gates over a set of actual input vectors.

B.4 Experimental Results

Selected LGsynth93 and ISCAS benchmark circuits were synthesized into the 45nm Nangate Library [Nangate 2008] using Synopsys DC^{TM}. For both the original and the protected circuits, the FIT rate was evaluated using equation B.1. For the purpose of evaluating the initial circuit-level FIT rate, 10,000 fault-injections were performed, to ensure small error bars. An arbitrary FIT rate (f_i), was assigned to each gate, proportional to the number of transistors in the gate.

We present the results as the FIT rate reduction which is the ratio of the protected FIT rate to that of the original circuit. It is important to note that this metric reflects the fact that failures can occur in the redundant logic and thus represents a net FIT rate reduction. If the original FIT rate was 10 and the FIT rate with the redundant logic is 5, the reduction is 2x.

The algorithm described in section B.3 was implemented using a Ruby script. The parameter N_{batch} was set between 20 and 100 based on the size of the circuit. N_{max} was set to 200 and N_{step} was set to 10. An 80% confidence interval was used ($z_{\alpha/2} = 1.28$) when evaluating the error bars. The cubes were generated with Espresso and the fault-injection simulations were performed with Mentor's Modelsim^{TM}. The redundant logic functions (F, H) produced by the script were synthesized with Synposys DC^{TM}.

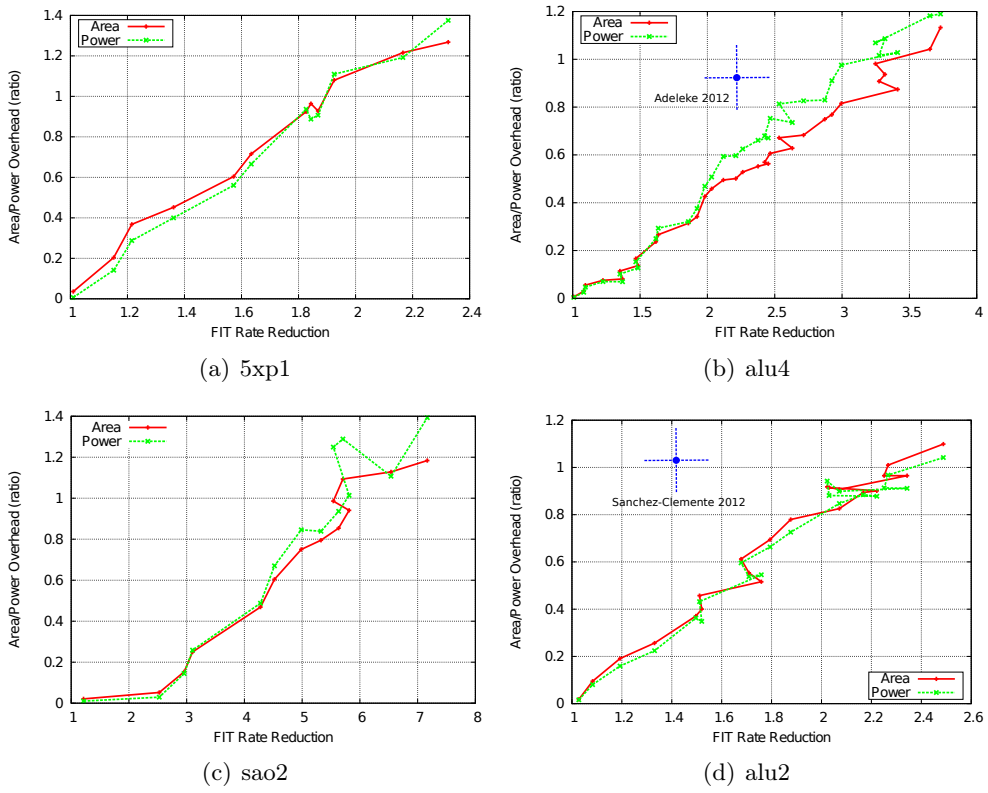
Table B.3: Summary of FIT Rate Improvement for Benchmark Circuits

Circuit	PI/PO	Num. Gates	Num. Cubes	FIT Reduction	Area (%)	Power (%)	Num. FIs
sao2	10/4	70	167	2x	5	3	13 270
alu4	14/8	469	2066	2x	46	51	358 410
5xp1	7/10	65	153	2x	100	111	13 160
ex5p	8/63	188	652	2x	69	49	106 130
f51m	8/8	73	155	2x	103	91	14 280
inc	7/9	69	142	2x	52	57	13 450
S386	13/13	64	163	2x	34	32	24 140
S1488	14/25	297	1639	2x	38	44	326 930
S1484	14/25	298	1636	2x	38	50	248 570

During synthesis, the tool was allowed to share logic between the F and H functions across multiple outputs but it was ensured there was no logic sharing with the original function G. We note that this approach is different from [Sanchez-Clemente 2012] where sharing between F and H was explicitly prevented during synthesis. When logic sharing is prevented, faults in one of the approximate functions are fully

blocked because the original function and the other approximation are correct (see figure B.1). Of course, this benefit comes with the downside that the area of the shared logic is increased. In future work, we intent to compare the results obtained with and without logic sharing between F and H.

In figure B.5 the area and power overhead required to achieve increasing levels of stuck-at fault masking for four different circuits are shown. Each data point was produced after 5 additional cubes were added to the redundant logic and the connecting lines show the order of cube selection. It is interesting to note that in some cases there are points where adding an additional cubes reduces the area and power, presumably because the extra cubes enables new logic simplification. Two data points are plotted comparing published results on the same circuits. We note however, that with our fault model, the per-gate failure rate varies, whereas in the previous work it was fixed.



(a) 5xp1 (b) alu4

(c) sao2 (d) alu2

Figure B.5: Area, Power Overhead versus FIT Rate Reduction

Table B.3 summarizes the area, power and timing overhead for ten circuits, showing the first generated result providing at least a 2x reduction in FIT rate. It is interesting to note that the required overhead to achieve a 2x reduction in the FIT rate varies from 5% to over 100%. We believe this is partly due to the SOP circuit representation, however, it suggests that, we may be far from the optimal solution.

B.5 Conclusions

Protecting random, combinatorial logic from faults is a challenging problem. The proposed approach based on a SOP representation, uses a fault model that accounts for different gates having different failure rates and shows promising results for selected, small circuits. It is true that the overheads are significant compared to those for protecting memories or other regular structures. However, given that the hard and soft FIT rates due to combinatorial logic are no longer trivial and that random logic represents a small fraction of most dies, they may be acceptable. In an industrial context, generic solutions are required which can be applied directly to a gate-level netlist, however this is not the case of many on-line test techniques. The proposed technique can be applied automatically to any combinatorial circuit and the number of fault injection simulations is carefully managed.

Additional work is underway to extend the fault model to include SETs and to account for the power overhead during the minimization process. In the implementation described here, the synthesis tool was allowed to share logic between the F and H functions. A study is underway to evaluate whether better results are obtained with or without such sharing. In addition, a method to decompose large circuits is also required in order to process industrial scale designs.

Appendix C

Protection of FPGA CRAM or TCAMs using BICS

C.1 Overview of SRAM Based FPGAs and TCAMs

FPGAs are programmable devices that can be configured to implement a specific logic function. The configuration information for an FPGA is stored in a memory called a Configuration RAM (CRAM) and there are three broad types of CRAMs : SRAM-based, flash-based and anti-fused based. Currently, SRAM-based FPGAs have significantly higher density and performance than flash or anti-fuse based FPGAs and there is significant interest in using them in aerospace applications. Unfortunately, the CRAM in SRAM-based FPGAs is sensitive to soft errors and upsets in the CRAM change the implemented logic function.

Clearly, not all bit-flips result in a change in the logic function and Xilinx classifies CRAM bits as shown in figure C.1. The *essential bits* are the ones that are actually used to control the logic in the design. Amongst these bits, the ones that control logic that the designer deems to be important are classified as *prioritized essential bits*. Finally, the *critical bits* are those which, when they are upset, actually cause a system-level error. This taxonomy is reproduced from [Xilinx 2012] in figure C.1.

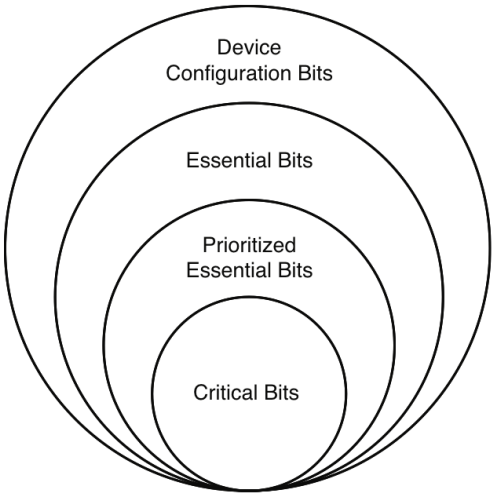


Figure C.1: Classification of CRAM Bits

Traditional ECC techniques are not easily applicable to the CRAM of an FPGA because all the bits of the entire memory are continuously read to control the logic array. A common technique to detect errors in the CRAM uses a scan engine. The scan engine cycles through the addresses of the CRAM, reading them and detecting whether there is an error. The detection can be performed using an error detecting or correcting code (e.g. parity or ECC). Typically the scan engine reads the CRAM in a linear pattern and the worst case error detection time can be on the order of several milliseconds for modern FPGAs. During the window of time between the occurrence of the upset and the detection of the error by the scan engine, the logic array is not performing the correct function. Even after the error in the CRAM is corrected, the effect of the incorrect logic function may persist. The smaller the window of time during which the fault is present, the less the risk that there will be a persistent system level effect. Scan-based techniques can be accelerated through the use of multiple, parallel scan engines, however, additional scan engines required additional area and power. In this section, we propose a technique that can significantly reduce the detection latency for SEUs in the CRAM of FPGAs with minimal area and power overhead.

C.2 Overview of BICS

A BICS is a circuit that can directly detect the current induced by the strike of an ionizing particle. BICS designs have been proposed for the detection of SEUs in memories [Vargas 1994, Gill 2005] as well as in logic [Zhang 2010, Zhang 2013]. An example BICS circuit is reproduced from [Wirth 2008] in figure C.2. Although these circuits are attractive for detecting Single Event Effects (SEEs), they suffer from two common shortcomings. First, a BICS circuit must be tuned so that it only detects current impulses that correspond to actual particle strikes. With increased process variability, this can be challenging. Second, BICS are subject to a high-rate of *false positives*. Frequently, a BICS will detect a current pulse that does not actually cause a bit-flip. These *false positives* may actually result in reduced availability, depending on the error recovery mechanism.

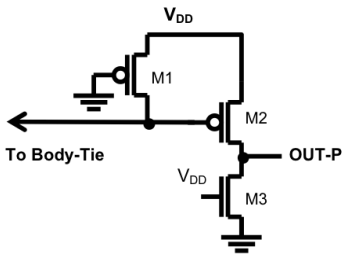


Figure C.2: Sample BICS Implementation

C.3 Use of BICs to Protect CRAMs

The idea proposed in this section consists of using a series of BICS circuits embedded in the CRAM of an FPGA to detect the ionizing particle when it strikes, as shown in figure C.3. Each BICS circuit detects upsets that correspond to a small range of logical addresses in the memory.

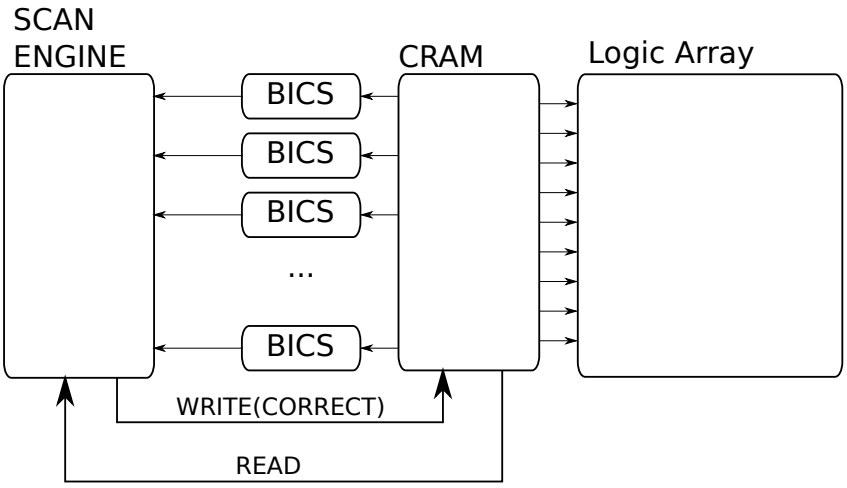


Figure C.3: Scan Engine Guided by BICS

When an ionizing event occurs, it is detected by one of the BICS and this is immediately communicated to the scan engine. Upon such an event, the scan engine interrupts its normal scan process and immediately starts scanning the small range of addresses within the array that are covered by the BICS that detected the strike. When the address with the error is located, it can be corrected immediately.

With the standard scan-based approach, the detection and correction interval depends on the time required by the scan engine to read the entire CRAM. With the proposed approach, the interval is bounded by the time for the scan engine to react to the BICS and then to read the small region of the CRAM that is covered by a single BICS. This time can be made small by increasing the number of BICS. The difference between the two approaches is illustrated in figure C.4.

BICS circuits must be tuned and they are prone to *false positives*. In the proposed application, such *false positives* are not a concern. If the BICS incorrectly interrupts the normal scan process and directs it to a region of the memory where there is no error, this is benign. At worst, this occasionally causes a small increase in the regular scan interval. The fact that the scan engine remains active ensures that if an ionizing event occurs that is *not* detected by the BICS, it will eventually be detected by the background scan.

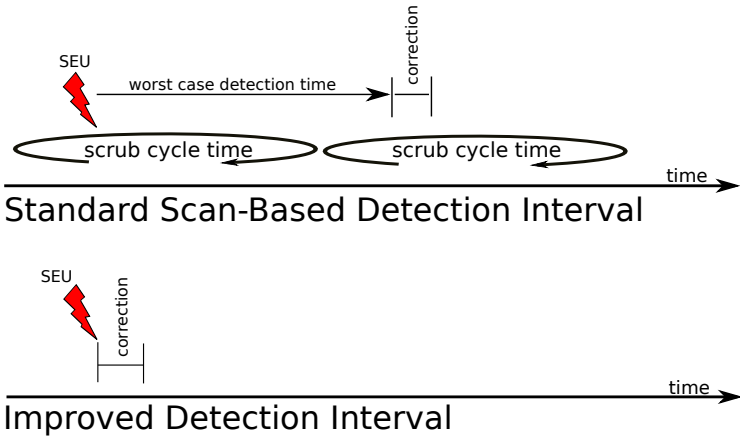


Figure C.4: Reduced Scan Interval due to BICS

C.4 Extension to TCAMs

A TCAM is a specialized memory which matches an incoming key against a set of stored entries where each entry contains a string of zeroes, ones or don't care bits. Storage cells in a TCAM are subject to soft errors and traditional ECC are not easily applicable because all the bits of the entire memory are continuously read to match against the incoming search key. A common technique to mitigate errors in TCAMs is to use a scan engine, conceptually similar to the type used for SRAM-based FPGAs. During the window of time between the occurrence of the error and the detection of the error by the scan, the TCAM may return incorrect responses. In networking applications, TCAMs are used for packet classification including the implementation of security features such as Access Control Lists. During the window of time between when an error occurs and when it is detected, packets may be mis-classified and potentially this could represent a security threat.

Exactly the same approach that is described for FPGA CRAM can be applied to TCAM. Specifically, when a BICS embedded in the TCAM array detects a potential ionizing radiation event, the existing scan engine is directed to scan the memory region where the event occurred. In this way, the average error detection time is greatly reduced.

C.5 Summary

The idea of using BICS to protect memories is quite old [Vargas 1994]. However, to the best of the author's knowledge, the idea of coupling a BICS circuit with a background scan engine in order to reduce error detection latency is new. The combination of these techniques is attractive because the *false positives* from the BICS are not an issue. At the same time, if there are events that are not detected by the BICS, they will be detected by the background scan.

Bibliography

[Adeleke 2012] A. Adeleke. Logic repair and soft error rate reduction. Master's thesis, Vamderbilt University, 2012. (Cited on page 139.)

[Aitken 2005] R. Aitken and B. Hold. *Modeling soft-error susceptibility for IP blocks.* In On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International, pages 70–73, July 2005. doi:10.1109/IOLTS.2005.44. (Cited on page 107.)

[Alexandrescu 2002] D. Alexandrescu, L. Anghel and M. Nicolaidis. *New methods for evaluating the impact of single event transients in VDSM ICs.* In Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on, pages 99 – 107, 2002. doi:10.1109/DFTVS.2002.1173506. (Cited on page 92.)

[Alexandrescu 2011] D. Alexandrescu, E. Costenaro and M. Nicolaidis. *A Practical Approach to Single Event Transients Analysis for Highly Complex Designs.* In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on, pages 155–163, 2011. doi:10.1109/DFT.2011.18. (Cited on page 93.)

[Alexandrescu 2012] D. Alexandrescu and E. Costenaro. *Towards optimized functional evaluation of SEE-induced failures in complex designs.* In On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, pages 182–187, 2012. doi:10.1109/IOLTS.2012.6313869. (Cited on pages 50 and 95.)

[Alexandrescu 2013] D. Alexandrescu, E. Costenaro and A. Evans. *State-aware single event analysis for sequential logic.* In On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International, pages 151–156, July 2013. doi:10.1109/IOLTS.2013.6604067. (Cited on pages 23 and 107.)

[Almukhaizim 2008] S. Almukhaizim and Y Makris. *Soft Error Mitigation Through Selective Addition of Functionally Redundant Wires.* Reliability, IEEE Transactions on, vol. 57, no. 1, pages 23–31, March 2008. doi:10.1109/TR.2008.916877. (Cited on page 140.)

[Antoni 2002] L. Antoni, R. Leveugle and B. Feher. *Using run-time reconfiguration for fault injection in hardware prototypes.* In Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on, pages 245–253, 2002. doi:10.1109/DFTVS.2002.1173521. (Cited on page 52.)

[Arlat 2003] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs and G.H. Leber. *Comparison of physical and software-implemented fault injection techniques.* Computers, IEEE Transactions on, vol. 52, no. 9, pages 1115–1133, Sept 2003. doi:10.1109/TC.2003.1228509. (Cited on pages 54 and 55.)

[Arlat 2011] J. Arlat and R. Moraes. *Collecting, Analyzing and Archiving Results from Fault Injection Experiments.* In Dependable Computing (LADC), 2011 5th Latin-American Symposium on, pages 100–105, April 2011. doi:10.1109/LADC.2011.19. (Cited on page 105.)

[Azizi 2006] N. Azizi and F.N. Najm. *A family of cells to reduce the soft-error-rate in ternary-CAM.* In Design Automation Conference, 2006 43rd ACM/IEEE, pages 779–784, 2006. doi:10.1109/DAC.2006.229229. (Cited on page 128.)

[Baarir 2009] S. Baarir, C. Braunstein, R. Clavel, E. Encrenaz, J.-M. Ilie, R. Leveugle, I. Mounier, L. Pierre and D. Poitrenaud. *Complementary Formal Approaches for Dependability Analysis.* In Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on, pages 331–339, oct. 2009. doi:10.1109/DFT.2009.21. (Cited on page 59.)

[Bahar 1993] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo and F. Somenzi. *Algebraic decision diagrams and their applications.* In Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on, pages 188–191, Nov 1993. doi:10.1109/ICCAD.1993.580054. (Cited on page 57.)

[Baumann 1995] R. Baumann, T. Hossain, S. Murata and I. Kitagawa. *Boron compounds as a dominant source of alpha particles in semiconductor devices.* In Reliability Physics Symposium, 1995. 33rd Annual Proceedings., IEEE International, pages 297–302, April 1995. doi:10.1109/RELPHY.1995.513695. (Cited on page 7.)

[Baumann 2002] R. Baumann. *The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction.* In Electron Devices Meeting, 2002. IEDM '02. International, pages 329–332, Dec 2002. doi:10.1109/IEDM.2002.1175845. (Cited on page 15.)

[Baumann 2005] R.C. Baumann. *Radiation-induced soft errors in advanced semiconductor technologies.* Device and Materials Reliability, IEEE Transactions on, vol. 5, no. 3, pages 305–316, 2005. doi:10.1109/TDMR.2005.853449. (Cited on pages 3 and 6.)

[Bayraktaroglu 2000] I. Bayraktaroglu and A. Orailoglu. *Unifying methodologies for high fault coverage concurrent and off-line test of digital filters.* In Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on, volume 2, pages 705–708 vol.2, 2000. doi:10.1109/ISCAS.2000.856426. (Cited on page 139.)

[Baze 2008] M.P. Baze, B. Hughlock, J. Wert, J. Tostenrude, L. Massengill, O. Amusan, R. Lacoe, K. Lilja and M. Johnson. *Angular Dependence of Single Event Sensitivity in Hardened Flip/Flop Designs.* Nuclear Science,

IEEE Transactions on, vol. 55, no. 6, pages 3295–3301, Dec 2008. doi:10.1109/TNS.2008.2009115. (Cited on page 22.)

[Berg 2013] M. Berg, K Label and J. Pellish. Microelectronics reliability and qualification working group meeting [online]. 2013. URL: https://nepp.nasa.gov/files/25702/2013MRQW_Berg_n272.pdf. (Cited on page 22.)

[Berrojo 2002a] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Lopez, M. Sonza Reorda and G. Squillero. *An industrial environment for high-level fault-tolerant structures insertion and validation.* In VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE, pages 229 – 236, 2002. doi:10.1109/VTS.2002.1011143. (Cited on page 51.)

[Berrojo 2002b] L. Berrojo, I. Gonzalez, F. Corno, M.S. Reorda, G. Squillero, L. Entrena and C. Lopez. *New techniques for speeding-up fault-injection campaigns.* In Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, pages 847 –852, 2002. doi:10.1109/DATE.2002.998398. (Cited on pages 64, 67 and 68.)

[Berrojo 2002c] L. Berrojo, I. Gonzalez, L. Entrena, C. Lopez, F. Corno, M. Sonza and G. Squillero. *Analysis of the equivalences and dominances of transient faults at the RT level.* In On-Line Testing Workshop, 2002. Proceedings of the Eighth IEEE International, page 193, 2002. doi:10.1109/OLT.2002.1030216. (Cited on page 51.)

[Bessot 1993] D. Bessot and R. Velazco. *Design of SEU-hardened CMOS memory cells: the HIT cell.* In Radiation and its Effects on Components and Systems, 1993.,RADECS 93., Second European Conference on, pages 563–570, 1993. doi:10.1109/RADECS.1993.316519. (Cited on page 20.)

[Bhaduri 2007] D. Bhaduri, S. Shukla, P. Graham and M. Gokhale. *Scalable techniques and tools for reliability analysis of large circuits.* In VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on, pages 705–710, Jan 2007. doi:10.1109/VLSID.2007.139. (Cited on page 57.)

[Bhuva 2011] B.L. Bhuva, K. Lilja, J. Holts, S-J Wen, R. Wong, S. Jagannathan, T.D. Loveless, M. McCurdy and Z. J. Diggins. *Comparative analysis of flip-flop designs for soft errors at advanced technology nodes.* In IC Design Technology (ICICDT), 2011 IEEE International Conference on, pages 1–4, 2011. doi:10.1109/ICICDT.2011.5783239. (Cited on page 76.)

[Binder 1975] D. Binder, E.C. Smith and A. B. Holman. *Satellite Anomalies from Galactic Cosmic Rays.* Nuclear Science, IEEE Transactions on, vol. 22, no. 6, pages 2675–2680, 1975. doi:10.1109/TNS.1975.4328188. (Cited on page 6.)

[Blaauw 2008] D. Blaauw, S. Kalaiselvan, K. Lai, Wei-Hsiang Ma, S. Pant, C. Tokunaga, S. Das and D. Bull. *Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance*. In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pages 400 –622, feb. 2008. doi:10.1109/ISSCC.2008.4523226. (Cited on page 29.)

[Bloom 1970] B. Bloom. *Space/Time Tradeoffs in Hash Coding with Allowable Errors*. Communication of ACM, vol. 13, no. 7, pages 422–426, 1970. (Cited on page 128.)

[Bowman 2008] K.A. Bowman, J.W. Tschanz, Nam Sung Kim, J.C. Lee, C.B. Wilkerson, S.-L.L. Lu, T. Karnik and V.K. De. *Energy-Efficient and Metastability-Immune Timing-Error Detection and Instruction-Replay-Based Recovery Circuits for Dynamic-Variation Tolerance*. In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pages 402 –623, feb. 2008. doi:10.1109/ISSCC.2008.4523227. (Cited on page 31.)

[Bryant 1986] R.E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. Computers, IEEE Transactions on, vol. C-35, no. 8, pages 677–691, Aug 1986. doi:10.1109/TC.1986.1676819. (Cited on page 57.)

[Cadence 1998] Cadence [online]. 1998. URL: http://www.kenmcmil.com/smv.html. (Cited on page 59.)

[Calin 1996] T. Calin, M. Nicolaidis and R. Velazco. *Upset hardened memory design for submicron CMOS technology*. Nuclear Science, IEEE Transactions on, vol. 43, no. 6, pages 2874 –2878, dec 1996. doi:10.1109/23.556880. (Cited on pages 20 and 22.)

[Cardarilli 2002] G.C. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli and R. Velazco. *Bit flip injection in processor-based architectures: a case study*. In On-Line Testing Workshop, 2002. Proceedings of the Eighth IEEE International, pages 117 – 127, 2002. doi:10.1109/OLT.2002.1030194. (Cited on page 54.)

[Cavrois 2008] V.F. Cavrois, V. Pouget, D. McMorrow, J.R. Schwank, N. Fel, F. Essely, R. S. Flores, P. Paillet, M. Gaillardin, D. Kobayashi, J. S. Melinger, O. Duhamel, P.E. Dodd and M.R. Shaneyfelt. *Investigation of the Propagation Induced Pulse Broadening (PIPB) Effect on Single Event Transients in SOI and Bulk Inverter Chains*. Nuclear Science, IEEE Transactions on, vol. 55, no. 6, pages 2842–2853, Dec 2008. doi:10.1109/TNS.2008.2007724. (Cited on pages 13 and 91.)

[Chang 1974] H.Y.-P. Chang, S.G. Chappell, C.H. Elmendorf and L.D. Schmidt. *Comparison of Parallel and Deductive Fault Simulation Methods*. Computers,

IEEE Transactions on, vol. C-23, no. 11, pages 1132–1138, Nov 1974. doi:10.1109/T-C.1974.223820. (Cited on page 50.)

[Chao 2002] H.J. Chao. *Next generation routers*. Proceedings of the IEEE, vol. 90, no. 9, pages 1518–1558, Sep 2002. doi:10.1109/JPROC.2002.802001. (Cited on page 127.)

[Chen 2011] L. Chen, F. Firouzi, S. Kiamehr and M.B. Tahoori. *Fast and Accurate Soft Error Rate Estimation at the RTL Level*. In GMM/CI/ITG-Fachtagung ZuE, 2011. (Cited on page 62.)

[Cho 2013] Hyungmin Cho, S. Mirkhani, Chen-Yong Cher, J.A. Abraham and S. Mitra. *Quantitative evaluation of soft error injection techniques for robust system design*. In Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE, pages 1–10, 2013. (Cited on page 55.)

[Choudhury 2008] M.R. Choudhury and K. Mohanram. *Approximate logic circuits for low overhead, non-intrusive concurrent error detection*. In Design, Automation and Test in Europe, 2008. DATE '08, pages 903–908, March 2008. doi:10.1109/DATE.2008.4484789. (Cited on page 140.)

[Choudhury 2013] M.R. Choudhury and K. Mohanram. *Low Cost Concurrent Error Masking Using Approximate Logic Circuits*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 32, no. 8, pages 1163–1176, Aug 2013. doi:10.1109/TCAD.2013.2250581. (Cited on pages 139 and 140.)

[Civera 2001] P. Civera, L. Macchiarulo, M. Rebaudengo, M.S. Reorda and M. Violante. *Exploiting circuit emulation for fast hardness evaluation*. Nuclear Science, IEEE Transactions on, vol. 48, no. 6, pages 2210–2216, Dec 2001. doi:10.1109/23.983197. (Cited on page 52.)

[Darbari 2008] A. Darbari, B. Al Hashimi, P. Harrod and D. Bradley. *A New Approach for Transient Fault Injection Using Symbolic Simulation*. In On-Line Testing Symposium, 2008. IOLTS '08. 14th IEEE International, pages 93–98, July 2008. doi:10.1109/IOLTS.2008.59. (Cited on page 59.)

[Das 2000] D. Das, N.A. Touba, M. Seuring and M. Gossel. *Low cost concurrent error detection based on module weight-based codes*. In On-Line Testing Workshop, 2000. Proceedings. 6th IEEE International, pages 171–176, 2000. doi:10.1109/OLT.2000.856633. (Cited on page 139.)

[Das 2009] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull and D.T. Blaauw. *RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance*. Solid-State Circuits, IEEE Journal of, vol. 44, no. 1, pages 32 –48, jan. 2009. doi:10.1109/JSSC.2008.2007145. (Cited on pages 29 and 31.)

[de Andres 2008] D. de Andres, J.C. Ruiz, D. Gil and P. Gil. *Fault Emulation for Dependability Evaluation of VLSI Systems*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 16, no. 4, pages 422 –431, april 2008. doi:10.1109/TVLSI.2008.917428. (Cited on page 51.)

[de Micheli 1994] G. de Micheli. Synthesis and optimization of digital circuits. Mc Graw Hill series in Electrical and Computer Engineering. Mc Graw Hill, 1994. (Cited on page 141.)

[Devadas 1996] S Devadas, A. Ghosh and K. Keutzer. *An observability-based code coverage metric for functional simulation*. In Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on, pages 418–425, 1996. doi:10.1109/ICCAD.1996.569832. (Cited on page 93.)

[Dharmapurikar 2006] Sarang Dharmapurikar, Praveen Krishnamurthy and D.E. Taylor. *Longest prefix matching using bloom filters*. Networking, IEEE/ACM Transactions on, vol. 14, no. 2, pages 397–409, April 2006. doi:10.1109/TNET.2006.872576. (Cited on page 131.)

[Dicello 1983] J.F. Dicello, C. W. McCabe, J.D. Doss and M. Paciotti. *The Relative Efficiency of Soft-Error Induction in 4K Static RAMS by Muons and Pions*. Nuclear Science, IEEE Transactions on, vol. 30, no. 6, pages 4613–4615, 1983. doi:10.1109/TNS.1983.4333182. (Cited on page 7.)

[Dixit 2011] A. Dixit and Alan Wood. *The impact of new technology on soft error rates*. In Reliability Physics Symposium (IRPS), 2011 IEEE International, pages 5B.4.1–5B.4.7, April 2011. doi:10.1109/IRPS.2011.5784522. (Cited on page 15.)

[Dutta 2008] A. Dutta and A. Jas. *Combinational Logic Circuit Protection Using Customized Error Detecting and Correcting Codes*. In Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on, pages 68–73, March 2008. doi:10.1109/ISQED.2008.4479700. (Cited on page 139.)

[Ebrahimi 2014] M. Ebrahimi, A. Evans, M.B. Tahoori, R. Seyyedi, E. Costenaro and D. Alexandrescu. *Comprehensive Analysis of Alpha and Neutron Particle-induced Soft Errors in an Embedded Processor at Nanoscales*. In Design, Automation Test in Europe Conference Exhibition (DATE), 2014, 2014. (Cited on pages 9, 15, 16 and 62.)

[Entrena 2009] L. Entrena, M.G. Valderas, R.F. Cardenal, M.P. Garcia and C.L. Ongil. *SET Emulation Considering Electrical Masking Effects*. Nuclear Science, IEEE Transactions on, vol. 56, no. 4, pages 2021–2025, Aug 2009. doi:10.1109/TNS.2009.2013346. (Cited on page 52.)

[Entrena 2012] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela and C. Lopez-Ongil. *Soft Error Sensitivity Evaluation*

of Microprocessors by Multilevel Emulation-Based Fault Injection. Computers, IEEE Transactions on, vol. 61, no. 3, pages 313–322, March 2012. doi:10.1109/TC.2010.262. (Cited on pages 53 and 62.)

[Ernst 2004] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim and K. Flautner. *Razor: circuit-level correction of timing errors for low-power operation.* Micro, IEEE, vol. 24, no. 6, pages 10 –20, nov.-dec. 2004. doi:10.1109/MM.2004.85. (Cited on page 27.)

[Evans 2012] A. Evans, M. Nicolaidis, Shi-Jie Wen, D. Alexandrescu and E. Costenaro. *RIIF - Reliability information interchange format.* In On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, pages 103–108, 2012. doi:10.1109/IOLTS.2012.6313849. (Cited on page 90.)

[Evans 2013] A. Evans, D. Alexandrescu, E. Costenaro and Liang Chen. *Hierarchical RTL-based combinatorial SER estimation.* In On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International, pages 139–144, 2013. doi:10.1109/IOLTS.2013.6604065. (Cited on pages 20, 89 and 125.)

[Everitt 1993] Brian Everitt. Cluster analysis. Halsted Press and John Wiley, 3rd edition édition, 1993. (Cited on page 81.)

[Fang 2013] Y. Fang and T. Oates. *Thermal Neutron Induced Soft-Errors in Advanced Memory and Logic Devices*, 2013. doi:10.1109/TDMR.2013.2287699. (Cited on page 6.)

[Fojtik 2012] M. Fojtik, D. Fick, Yejoong Kim, N. Pinckney, D. Harris, D. Blaauw and D Sylvester. *Bubble Razor: An architecture-independent approach to timing-error detection and correction.* In Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International, pages 488–490, Feb 2012. doi:10.1109/ISSCC.2012.6177103. (Cited on pages 32 and 33.)

[Fojtik 2013] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D.M. Harris, D. Blaauw and D. Sylvester. *Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction.* Solid-State Circuits, IEEE Journal of, vol. 48, no. 1, pages 66–81, Jan 2013. doi:10.1109/JSSC.2012.2220912. (Cited on pages 32 and 33.)

[Fowler 2011] M. Fowler and R. Parsons. Domain-specific modeling languages. Pearson Education. Addison-Wesley Publishing Company, 2011. (Cited on page 106.)

[Fujita 2014] Tomohiro Fujita, SinNyoung Kim and Hidetoshi Onodera. *Computer simulation of radiation-induced clock-perturbation in phase-locked loop with analog behavioral model.* In Quality Electronic Design (ISQED), 2014 15th

International Symposium on, pages 230–235, March 2014. doi:10.1109/ISQED.2014.6783330. (Cited on page 8.)

[George 2010] N.J. George, C.R. Elks, B.W. Johnson and J. Lach. *Transient fault models and AVF estimation revisited*. In Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on, pages 477–486, June 2010. doi:10.1109/DSN.2010.5544276. (Cited on page 57.)

[Geppert 1991] L.M. Geppert, U. Bapst, D.F. Heidel and K.A. Jenkins. *Ion microbeam probing of sense amplifiers to analyze single event upsets in a CMOS DRAM*. Solid-State Circuits, IEEE Journal of, vol. 26, no. 2, pages 132–134, Feb 1991. doi:10.1109/4.68127. (Cited on page 7.)

[Gertsbakh 1989] I.B. Gertsbakh. Statistical reliability theory. Probability: Pure and Applied. Marcel Dekker Inc., 1989. (Cited on page 110.)

[Gill 2005] B. Gill, M. Nicolaidis, F. Wolff, C. Papachristou and S. Garverick. *An efficient BICS design for SEUs detection and correction in semiconductor memories*. In Design, Automation and Test in Europe, 2005. Proceedings, pages 592–597 Vol. 1, March 2005. doi:10.1109/DATE.2005.54. (Cited on page 150.)

[Gill 2009] B. Gill, N. Seifert and V. Zia. *Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node*. In Reliability Physics Symposium, 2009 IEEE International, pages 199–205, 2009. doi:10.1109/IRPS.2009.5173251. (Cited on page 12.)

[Gordon 2004] M.S. Gordon, P. Goldhagen, K.P. Rodbell, T.H. Zabel, H. H K Tang, J. M. Clem and P. Bailey. *Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground*. Nuclear Science, IEEE Transactions on, vol. 51, no. 6, pages 3427–3434, 2004. doi:10.1109/TNS.2004.839134. (Cited on page 4.)

[Gordon 2005] M.S. Gordon, P. Goldhagen, K.P. Rodbell, T.H. Zabel, H.H.K. Tang, J.M. Clem and P. Bailey. *Correction to Measurement of the Flux and Energy Spectrum of Cosmic-Ray Induced Neutrons on the Ground*. Nuclear Science, IEEE Transactions on, vol. 52, no. 6, pages 2703–2703, 2005. doi:10.1109/TNS.2005.860694. (Cited on page 4.)

[Gracia 2008] J. Gracia, L. Saiz, J.-C. Baraza, D. Gil and P. Gil. *Analysis of the influence of intermittent faults in a microcontroller*. In Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on, pages 1–6, April 2008. doi:10.1109/DDECS.2008.4538761. (Cited on page 93.)

[Guenzer 1979] C.S. Guenzer, E.A. Wolicki and R.G. Allas. *Single Event Upset of Dynamic Rams by Neutrons and Protons*. Nuclear Science, IEEE Transac-

tions on, vol. 26, no. 6, pages 5048–5052, 1979. doi:10.1109/TNS.1979.4330270. (Cited on page 6.)

[Hayes 2007] J.P. Hayes, I. Polian and B. Becker. *An Analysis Framework for Transient-Error Tolerance*. In VLSI Test Symposium, 2007. 25th IEEE, pages 249–255, May 2007. doi:10.1109/VTS.2007.13. (Cited on page 60.)

[Hazucha 2003] P. Hazucha, T. Karnik, S. Walstra, B. Bloechel, J. Tschanz, J. Maiz, K. Soumyanath, G. Dermer, S. Narendra, V. De and S. Borkar. *Measurements and analysis of SER tolerant latch in a 90 nm dual-Vt CMOS process*. In Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003, pages 617–620, Sept 2003. doi:10.1109/CICC.2003.1249472. (Cited on pages 22, 23 and 42.)

[Heald 2005] Heald. *How Cosmic Rays Cause Computer Down Time*, March 2005. URL: http://www.ewh.ieee.org/r6/scv/rl/articles/ser-050323-talk-ref.pdf [accessedJanuary 2014]. (Cited on page 4.)

[Heidel 1993] D.F. Heidel, U.H. Bapst, K.A. Jenkins, L.M. Geppert and T. H. Zabel. *Ion microbeam radiation system*. Nuclear Science, IEEE Transactions on, vol. 40, no. 2, pages 127–134, Apr 1993. doi:10.1109/23.212328. (Cited on page 7.)

[Ibe 2010] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba. *Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule*. Electron Devices, IEEE Transactions on, vol. 57, no. 7, pages 1527–1538, 2010. doi:10.1109/TED.2010.2047907. (Cited on page 15.)

[Ibe 2012] E. Ibe, T. Toba, K. Shimbo and H. Taniguchi. *Fault-based reliable design-on-upper-bound of electronic systems for terrestrial radiation including muons, electrons, protons and low energy neutrons*. In On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, pages 49–54, 2012. doi:10.1109/IOLTS.2012.6313840. (Cited on pages 3 and 16.)

[Jain 1985] S.K. Jain and V.D. Agrawal. *Statistical Fault Analysis*. Design Test of Computers, IEEE, vol. 2, no. 1, pages 38–44, Feb 1985. doi:10.1109/MDT.1985.294683. (Cited on page 60.)

[JEDEC 2006] JEDEC. *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, October 2006. URL: http://www.jedec.org/sites/default/files/docs/jesd89a.pdf. (Cited on pages 4, 6 and 111.)

[JEDEC 2010] JEDEC. *Failure Mechanisms and Models for Semiconductor Devices (JEP122G)*, 2010. URL: http://www.jedec.org/sites/default/files/docs/JEP122F.pdf. (Cited on pages 112 and 126.)

[Johnson 2000] Richard Johnson. Probability and statistics for engineers. Prentice Hall Inc., 6th edition édition, 2000. (Cited on page 72.)

[Kolasinski 1979] W.A. Kolasinski, J. B. Blake, J. K. Anthony, W.E. Price and E.C. Smith. *Simulation of Cosmic-Ray Induced Soft Errors and Latchup in Integrated-Circuit Computer Memories*. Nuclear Science, IEEE Transactions on, vol. 26, no. 6, pages 5087–5091, 1979. doi:10.1109/TNS.1979.4330278. (Cited on page 6.)

[Krishnan 2009] S.C. Krishnan, R. Panigrahy and S. Parthasarathy. *Error-Correcting Codes for Ternary Content Addressable Memories*. Computers, IEEE Transactions on, vol. 58, no. 2, pages 275–279, Feb 2009. doi:10.1109/TC.2008.179. (Cited on pages 127 and 128.)

[Lakshminarayanan 2005] Karthik Lakshminarayanan, Anand Rangarajan and Srinivasan Venkatachary. *Algorithms for advanced packet classification with ternary CAMs*. In In ACM SIGCOMM, pages 193–204, 2005. (Cited on pages 131 and 134.)

[Lee 2010] Hsiao-Heng Kelin Lee, K. Lilja, M. Bounasser, P. Relangi, I.R. Linscott, U.S. Inan and S. Mitra. *LEAP: Layout Design through Error-Aware Transistor Positioning for soft-error resilient sequential cell design*. In Reliability Physics Symposium (IRPS), 2010 IEEE International, pages 203–212, 2010. doi:10.1109/IRPS.2010.5488829. (Cited on pages 20, 40 and 42.)

[Leveugle 2002] R. Leveugle. *Automatic modifications of high level VHDL descriptions for fault detection or tolerance*. In Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, pages 837–841, 2002. doi:10.1109/DATE.2002.998396. (Cited on page 139.)

[Leveugle 2009] R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert. *Statistical fault injection: Quantified error and confidence*. In Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09., pages 502 –506, april 2009. (Cited on page 72.)

[Li 2005] Xiaodong Li, S.V. Adve, P. Bose and J.A. Rivers. *SoftArch: an architecture-level tool for modeling and analyzing soft errors*. In Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on, pages 496–505, 2005. doi:10.1109/DSN.2005.88. (Cited on page 93.)

[Li 2008] Y Li, S. Makar and S Mitra. *CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns*. In Design, Automation and Test in Europe, 2008. DATE '08, pages 885–890, March 2008. doi:10.1109/DATE.2008.4484786. (Cited on page 139.)

[Limbrick 2011] D.B. Limbrick, D.A. Black, K. Dick, N.M. Atkinson, N.J. Gaspard, J.D. Black, W.H. Robinson and A.F. Witulski. *Impact of logic synthesis*

on soft error vulnerability using a 90-nm bulk CMOS digital cell library. In Southeastcon, 2011 Proceedings of IEEE, pages 430–434, 2011. doi:10.1109/SECON.2011.5752980. (Cited on page 91.)

[Lin 2012] Hsiu-Yi Lin, Chun-Yao Wang, Shih-Chieh Chang, Yung-Chih Chen, Hsuan-Ming Chou, Ching-Yi Huang, Yen-Chi Yang and Chun-Chien Shen. *A probabilistic analysis method for functional qualification under Mutation Analysis.* In Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pages 147–152, 2012. doi:10.1109/DATE.2012.6176448. (Cited on page 93.)

[Lopez-Ongil 2007] Celia Lopez-Ongil, Mario Garcia-Valderas, Marta Portela-Garcia and Luis Entrena. *Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation.* Nuclear Science, IEEE Transactions on, vol. 54, no. 1, pages 252 –261, feb. 2007. doi:10.1109/TNS.2006.889115. (Cited on pages 51 and 52.)

[Mahatme 2011] N.N. Mahatme, S. Jagannathan, T.D. Loveless, L.W. Massengill, B.L. Bhuva, S.-J. Wen and R. Wong. *Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process.* Nuclear Science, IEEE Transactions on, vol. 58, no. 6, pages 2719 –2725, dec. 2011. doi:10.1109/TNS.2011.2171993. (Cited on pages 12, 16 and 20.)

[May 1978] Timothy C. May and Murray H. Woods. *A New Physical Mechanism for Soft Errors in Dynamic Memories.* In Reliability Physics Symposium, 1978. 16th Annual, pages 33–40, 1978. doi:10.1109/IRPS.1978.362815. (Cited on page 6.)

[May 1979] T.C. May and Murray H. Woods. *Alpha-particle-induced soft errors in dynamic memories.* Electron Devices, IEEE Transactions on, vol. 26, no. 1, pages 2–9, 1979. doi:10.1109/T-ED.1979.19370. (Cited on page 6.)

[Meyer 1997] Bertrand Meyer. Object-oriented software construction. Prentice Hall Inc., 1997. (Cited on page 117.)

[Miskov-Zivanov 2006] N. Miskov-Zivanov and D. Marculescu. *MARS-C: modeling and reduction of soft errors in combinational circuits.* In Design Automation Conference, 2006 43rd ACM/IEEE, pages 767–772, 2006. doi:10.1109/DAC.2006.229323. (Cited on pages 57, 58 and 92.)

[Miskov-Zivanov 2010] Natasa Miskov-Zivanov and D. Marculescu. *Formal modeling and reasoning for reliability analysis.* In Design Automation Conference (DAC), 2010 47th ACM/IEEE, pages 531–536, 2010. (Cited on pages 15, 57 and 96.)

[Mitra 2007] S. Mitra, Ming Zhang, N. Seifert, T.M. Mak and Kee Sup Kim. *Built-In Soft Error Resilience for Robust System Design.* In Integrated Circuit

Design and Technology, 2007. ICICDT '07. IEEE International Conference on, pages 1 –6, 30 2007-june 1 2007. doi:10.1109/ICICDT.2007.4299587. (Cited on page 26.)

[Mohammadi 2012] A. Mohammadi, M. Ebrahimi, A. Ejlali and S.-G. Miremadi. *SCFIT: A FPGA-based fault injection technique for SEU fault model.* In Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pages 586–589, March 2012. doi:10.1109/DATE.2012.6176538. (Cited on page 51.)

[Mohanram 2003] K. Mohanram and N.A. Touba. *Partial error masking to reduce soft error failure rate in logic circuits.* In Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on, pages 433–440, Nov 2003. doi:10.1109/DFTVS.2003.1250141. (Cited on page 139.)

[Moore 1965] Gordon Moore. *Cramming more components onto integrated circuits.* Electronics, vol. 38, page 114, 1965. (Cited on page 16.)

[Mukherjee 2003] S.S. Mukherjee, C.T. Weaver, J. Emer, S.K. Reinhardt and T. Austin. *Measuring architectural vulnerability factors.* Micro, IEEE, vol. 23, no. 6, pages 70–75, 2003. doi:10.1109/MM.2003.1261389. (Cited on page 56.)

[Mukherjee 2005] S.S. Mukherjee, J. Emer and S.K. Reinhardt. *The soft error problem: an architectural perspective.* In High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on, pages 243–247, 2005. doi:10.1109/HPCA.2005.37. (Cited on page 93.)

[Mukherjee 2008] S.S. Mukherjee. Architecture design for soft errors. Morgan Kaufmann, 2008. (Cited on pages 47 and 56.)

[Naffziger 2005] S. Naffziger, T. Grutkowski and B. Stackhouse. *The implementation of a 2-core, multi-threaded Itanium reg; family processor.* In Integrated Circuit Design and Technology, 2005. ICICDT 2005. 2005 International Conference on, pages 43–48, 2005. doi:10.1109/ICICDT.2005.1502587. (Cited on page 75.)

[Nangate 2008] Nangate. *The NanGate 45nm Open Cell Library*, 2008. URL: http://www.nangate.com/?page_id=22. (Cited on pages 41 and 145.)

[Nguyen 2005] H.T. Nguyen, Y. Yagil, N. Seifert and M. Reitsma. *Chip-level soft error estimation method.* Device and Materials Reliability, IEEE Transactions on, vol. 5, no. 3, pages 365 – 381, sept. 2005. doi:10.1109/TDMR.2005.858334. (Cited on pages 12, 15, 63 and 95.)

[Nicolaidis 1997] M. Nicolaidis, R.O. Duarte, S. Manich and J. Figueras. *Fault-secure parity prediction arithmetic operators.* Design Test of Computers,

IEEE, vol. 14, no. 2, pages 60–71, Apr 1997. doi:10.1109/54.587743. (Cited on page 139.)

[Nicolaidis 1998] M. Nicolaidis and R.O. Duarte. *Design of fault-secure parity-prediction Booth multipliers*. In Design, Automation and Test in Europe, 1998., Proceedings, pages 7–14, Feb 1998. doi:10.1109/DATE.1998.655830. (Cited on page 139.)

[Nicolaidis 1999] M. Nicolaidis. *Time redundancy based soft-error tolerance to rescue nanometer technologies*. In VLSI Test Symposium, 1999. Proceedings. 17th IEEE, pages 86–94, 1999. doi:10.1109/VTEST.1999.766651. (Cited on pages 21 and 27.)

[Nicolaidis 2007a] M. Nicolaidis. *GRAAL: A Fault-Tolerant Architecture for Enabling Nanometric Technologies*. In On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International, page 255, july 2007. doi:10.1109/IOLTS.2007.35. (Cited on page 24.)

[Nicolaidis 2007b] M. Nicolaidis. *GRAAL: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies*. In Test Conference, 2007. ITC 2007. IEEE International, pages 1 –10, oct. 2007. doi:10.1109/TEST.2007.4437666. (Cited on page 24.)

[Nicolaidis 2011] Michael Nicolaidis, editor. Soft errors in modern electronic systems. Volume 41 of Frontiers in Electronic Testing. Spring Verlag, 2011. (Cited on pages 6 and 11.)

[Nicolaidis 2013] Michael Nicolaidis, Said Hamdioui, Dimitris Gizopoulos, Groeseneken Guido, Arnaud Grasset and Philippe Bonnot. *Reliability challenges of real-time systems in forthcoming technology nodes*. In Design, Automation Test in Europe Conference Exhibition (DATE), 2013. (Cited on page 39.)

[Normand 1996] E. Normand. *Single-event effects in avionics*. Nuclear Science, IEEE Transactions on, vol. 43, no. 2, pages 461–474, 1996. doi:10.1109/23.490893. (Cited on page 4.)

[Normand 2010] E. Normand, J.L. Wert, H. Quinn, T.D. Fairbanks, S. Michalak, G. Grider, P. Iwanchuk, J. Morrison, S. Wender and S. Johnson. *First Record of Single-Event Upset on Ground, Cray-1 Computer at Los Alamos in 1976*. Nuclear Science, IEEE Transactions on, vol. 57, no. 6, pages 3114–3120, 2010. doi:10.1109/TNS.2010.2083687. (Cited on page 6.)

[Omana 2003] M. Omana, G. Papasso, D. Rossi and C. Metra. *A model for transient fault propagation in combinatorial logic*. In On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE, pages 111–115, July 2003. doi:10.1109/OLT.2003.1214376. (Cited on page 58.)

[Opencores 2013] Opencores. *Open Cores Web Page*, 2013. URL: http://www.opencores.org [accessed06/01/2013]. (Cited on page 77.)

[OpenRisc 2013] OpenRisc. *Open Cores Open Risc Project*, 2013. URL: http://opencores.org/or1k/Main_Page [accessedJanuary 2013]. (Cited on page 96.)

[Polian 2008] I. Polian, S.M. Reddy and B. Becker. *Scalable Calculation of Logical Masking Effects for Selective Hardening Against Soft Errors*. In Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual, pages 257 –262, april 2008. doi:10.1109/ISVLSI.2008.22. (Cited on pages 60 and 92.)

[Pontarelli 2010] S. Pontarelli, M. Ottavi and A. Salsano. *Error Detection and Correction in Content Addressable Memories*. In Defect and Fault Tolerance in VLSI Systems (DFT), 2010 IEEE 25th International Symposium on, pages 420–428, Oct 2010. doi:10.1109/DFT.2010.56. (Cited on page 134.)

[Pontarelli 2013a] S. Pontarelli and M. Ottavi. *Error Detection and Correction in Content Addressable Memories by Using Bloom Filters*. Computers, IEEE Transactions on, vol. 62, no. 6, pages 1111–1126, June 2013. doi:10.1109/TC.2012.56. (Cited on page 128.)

[Pontarelli 2013b] Salvatore Pontarelli, Marco Ottavi, Adrian Evans and Shi-Jie Wen. *Error detection in Ternary CAMs using Bloom Filters*. In Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pages 1474–1479, 2013. doi:10.7873/DATE.2013.300. (Cited on page 127.)

[Rajaramant 2006] R. Rajaramant, J.S. Kim, N. Vijaykrishnan, Y. Xie and M.J. Irwin. *SEAT-LA: a soft error analysis tool for combinational logic*. In VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on, 2006. doi:10.1109/VLSID.2006.143. (Cited on page 92.)

[Ramakrishnan 2008] K. Ramakrishnan, R. Rajaramant, N. Vijaykrishnan, Y. Xie, M.J. Irwin and K. Unlu. *Hierarchical Soft Error Estimation Tool (HSEET)*. In Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on, pages 680–683, 2008. doi:10.1109/ISQED.2008.4479819. (Cited on page 92.)

[Ramanarayanan 2009] R. Ramanarayanan, V. Degalahal, R. Krishnan, Jungsub Kim, V. Narayanan, Yuan Xie, M.J. Irwin and K. Unlu. *Modeling Soft Errors at the Device and Logic Levels for Combinational Circuits*. Dependable and Secure Computing, IEEE Transactions on, vol. 6, no. 3, pages 202–216, 2009. doi:10.1109/TDSC.2007.70231. (Cited on page 93.)

[Rao 2006] R.R. Rao, K. Chopra, D. Blaauw and D Sylvester. *An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits*.

In Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, volume 1, pages 1–6, 2006. doi:10.1109/DATE.2006.244060. (Cited on page 93.)

[Ren 2014] Y. Ren, A.-L. He, S.-T. Shi, G. Guo, L. Chen, S.-J. Wen, R. Wong, N. W. van Vonno and B. L. Bhuva. *Single-Event Transient Measurements on a DC/DC Pulse Width Modulator Using Heavy Ion, Proton, and Pulsed Laser*. Journal of Electronic Testing, January 2014. (Cited on page 9.)

[Rennie 2012] D. Rennie, D. Li, M. Sachdev, B.L. Bhuva, S. Jagannathan, ShiJie Wen and R. Wong. *Performance, Metastability, and Soft-Error Robustness Trade-offs for Flip-Flops in 40 nm CMOS*. Circuits and Systems I: Regular Papers, IEEE Transactions on, vol. 59, no. 8, pages 1626–1634, Aug 2012. doi:10.1109/TCSI.2012.2206505. (Cited on page 22.)

[Roche 2013] P. Roche, J.-L. Autran, G. Gasiot and D. Munteanu. *Technology downscaling worsening radiation effects in bulk: SOI to the rescue*. In Electron Devices Meeting (IEDM), 2013 IEEE International, pages 31.1.1–31.1.4, Dec 2013. doi:10.1109/IEDM.2013.6724728. (Cited on page 15.)

[Rudell 1987] R.L. Rudell and A. Sangiovanni-Vincentelli. *Multiple-Valued Minimization for PLA Optimization*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 6, no. 5, pages 727–750, September 1987. doi:10.1109/TCAD.1987.1270318. (Cited on page 141.)

[Sai-Halasz 1983] G.A. Sai-Halasz. *Cosmic ray induced soft error rate in VLSI circuits*. Electron Device Letters, IEEE, vol. 4, no. 6, pages 172–174, 1983. doi:10.1109/EDL.1983.25694. (Cited on page 7.)

[Saleh 1990] Abdallah M. Saleh, Juan J. Serrano and Janak H. Patel. *Reliability of scrubbing recovery-techniques for memory systems*. IEEE Transactions on Reliability, vol. 39, no. 1, pages 114–122, 1990. (Cited on page 118.)

[Sanchez-Clemente 2012] A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas and C. Lopez-Ongil. *Logic masking for SET Mitigation Using Approximate Logic Circuits*. In On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, pages 176–181, June 2012. doi:10.1109/IOLTS.2012.6313868. (Cited on pages 139, 140 and 145.)

[Sanchez-Macian 2013] Alfonso Sanchez-Macian, Pedro Reviriego and Juan Antonio Maestro. *Modeling Reliability of Memories Protected with Error Correction Codes with RIIF*. In RIIF DATE 2013 Workshop: Towards Standards for Specifying and Modelling the Reliability of Complex Electronic Systems, 2013. (Cited on page 118.)

[Seifert 2004] N. Seifert and N. Tam. *Timing vulnerability factors of sequentials*. Device and Materials Reliability, IEEE Transactions on, vol. 4, no. 3, pages

516 – 522, sept. 2004. doi:10.1109/TDMR.2004.831993. (Cited on pages 11 and 12.)

[Seifert 2005] N. Seifert, P. Shipley, M.D. Pant, V. Ambrose and B. GiII. *Radiation-induced clock jitter and race.* In Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International, pages 215 – 222, 17-21, 2005. doi:10.1109/RELPHY.2005.1493087. (Cited on pages 8 and 23.)

[Seifert 2006] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill and J. Maiz. *Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices.* In Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International, pages 217 –225, march 2006. doi:10.1109/RELPHY.2006.251220. (Cited on page 112.)

[Seifert 2010a] N. Seifert. Radiation-induced soft errors - a chip-level modeling persepective. Foundations and Trends in Electronic Design Automation. now Publishers Inc., 2010. (Cited on pages 4, 23 and 26.)

[Seifert 2010b] N. Seifert, V. Ambrose, B. Gill, Q. Shi, R. Allmon, C. Recchia, S. Mukherjee, N. Nassif, J. Krause, J. Pickholtz and A. Balasubramanian. *On the radiation-induced soft error performance of hardened sequential elements in advanced bulk CMOS technologies.* In Reliability Physics Symposium (IRPS), 2010 IEEE International, pages 188 –197, may 2010. doi:10.1109/IRPS.2010.5488831. (Cited on pages 22 and 23.)

[Seifert 2012] N. Seifert, B. Gill, S. Jahinuzzaman, J. Basile, V. Ambrose, Quan Shi, R. Allmon and A. Bramnik. *Soft Error Susceptibilities of 22 nm Tri-Gate Devices.* Nuclear Science, IEEE Transactions on, vol. 59, no. 6, pages 2666–2673, Dec 2012. doi:10.1109/TNS.2012.2218128. (Cited on page 15.)

[Seshia 2007] S.A. Seshia, Wenchao Li and S. Mitra. *Verification-Guided Soft Error Resilience.* In Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07, pages 1 –6, april 2007. doi:10.1109/DATE.2007.364501. (Cited on pages 59 and 62.)

[Shazli 2008] S.Z. Shazli and M.B. Tahoori. *Obtaining Microprocessor Vulnerability Factor Using Formal Methods.* In Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on, pages 63–71, Oct 2008. doi:10.1109/DFT.2008.52. (Cited on page 59.)

[Shivakumar 2002] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger and L. Alvisi. *Modeling the effect of technology trends on the soft error rate of combinational logic.* In Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, pages 389–398, 2002. doi:10.1109/DSN.2002.1028924. (Cited on pages 15 and 16.)

[Shuler 2009] Robert Shuler. *Pulse Distortion in SET Measurements from Layout and Adjacent Signals*. NASA, 2009. URL: http://mc1soft.com/papers/2009_PulseDistortion.pdf. (Cited on page 7.)

[Sierawski 2006] B. D. Sierawski, B. L. Bhuva and L. W. Massengill. *Reducing Soft Error Rate in Logic Circuits Through Approximate Logic Functions*. Nuclear Science, IEEE Transactions on, vol. 53, no. 6, pages 3417 –3421, dec. 2006. doi:10.1109/TNS.2006.884352. (Cited on pages 139 and 140.)

[Sierawski 2011] B.D. Sierawski, R.A. Reed, M.H. Mendenhall, R.A. Weller, R.D. Schrimpf, Shi-Jie Wen, R. Wong, N. Tam and R.C. Baumann. *Effects of scaling on muon-induced soft errors*. In Reliability Physics Symposium (IRPS), 2011 IEEE International, pages 3C.3.1–3C.3.6, 2011. doi:10.1109/IRPS.2011.5784484. (Cited on pages 3 and 16.)

[Silburt 2008] A.L. Silburt, A. Evans, A. Burghelea, Shi-Jie Wen, D. Ward, R. Norrish and D. Hogle. *Specification and Verification of Soft Error Performance in Reliable Internet Core Routers*. Nuclear Science, IEEE Transactions on, vol. 55, no. 4, pages 2389 –2398, aug. 2008. doi:10.1109/TNS.2008.2001742. (Cited on page 47.)

[Silburt 2009] A.L. Silburt, A. Evans, I. Perryman, Shi-Jie Wen and D. Alexandrescu. *Design for Soft Error Resiliency in Internet Core Routers*. Nuclear Science, IEEE Transactions on, vol. 56, no. 6, pages 3551 –3555, dec. 2009. doi:10.1109/TNS.2009.2033915. (Cited on pages 47 and 70.)

[Slayman 2011] C. Slayman. *Soft error trends and mitigation techniques in memory devices*. In Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual, pages 1 –5, jan. 2011. doi:10.1109/RAMS.2011.5754515. (Cited on page 15.)

[Sondon 2013] S. Sondon, A. Falcon, P. Mandolesi, P. Julian, N. Vega, F. Nesprias, J. Davidson, F. Palumbo and M. Debray. *Diagnose of radiation induced single event effects in a PLL using a heavy ion microbeam*. In Test Workshop (LATW), 2013 14th Latin American, pages 1–5, April 2013. doi:10.1109/LATW.2013.6562682. (Cited on page 8.)

[Sridharan 2009] V. Sridharan and D.R. Kaeli. *Eliminating microarchitectural dependency from Architectural Vulnerability*. In High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on, pages 117–128, Feb 2009. doi:10.1109/HPCA.2009.4798243. (Cited on page 56.)

[Stackhouse 2008] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski and C. Lyles. *A 65nm 2-Billion-Transistor Quad-Core ItaniumÂ® Processor*. In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical

Papers. IEEE International, pages 92–598, 2008. doi:10.1109/ISSCC.2008.4523072. (Cited on page 75.)

[Stott 1998] D.T. Stott, G. Ries, Mei-Chen Hsueh and R.K. Iyer. *Dependability analysis of a high-speed network using software-implemented fault injection and simulated fault injection.* Computers, IEEE Transactions on, vol. 47, no. 1, pages 108–119, Jan 1998. doi:10.1109/12.656094. (Cited on page 54.)

[Synopsys 2006] Synopsys. *CCS Timing Liberty Syntax*, 6 2006. URL: ftp://ftp.synopsys.com/doc/Timing_Format_1.2/ccs_timing_format_1.2.pdf. (Cited on page 107.)

[Tang 2004] H. H K Tang and E.H. Cannon. *SEMM-2: a modeling system for single event upset analysis.* Nuclear Science, IEEE Transactions on, vol. 51, no. 6, pages 3342–3348, 2004. doi:10.1109/TNS.2004.839507. (Cited on page 7.)

[Telecordia 2011] Telecordia. Telcordia reliability software tool [online]. 2011. URL: http://telecom-info.telcordia.com/site-cgi/ido/docs2.pl?ID=011898615&page=reliability_software. (Cited on page 108.)

[Trivedi 1982] Kishor S. Trivedi. Probability and statistics with reliability queuing and computer science applications. Prentice Hall Inc., 1982. (Cited on page 72.)

[Valadimas 2010] S. Valadimas, Y. Tsiatouhas and A. Arapoyanni. *Timing error tolerance in nanometer ICs.* In On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International, pages 283 –288, july 2010. doi:10.1109/IOLTS.2010.5560189. (Cited on page 35.)

[Valadimas 2012] S. Valadimas, Y. Tsiatouhas, A. Arapoyanni and A. Evans. *Single event upset tolerance in flip-flop based microprocessor cores.* In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on, pages 79–84, 2012. doi:10.1109/DFT.2012.6378204. (Cited on page 36.)

[Valderas 2007] M.G. Valderas, P.. Peronnard, C. Lopez Ongil, R.. Ecoffet, F. Bezerra and R.. Velazco. *Two Complementary Approaches for Studying the Effects of SEUs on Digital Processors.* Nuclear Science, IEEE Transactions on, vol. 54, no. 4, pages 924 –928, aug. 2007. doi:10.1109/TNS.2007.893871. (Cited on pages 51 and 53.)

[Valderas 2010] M.G. Valderas, M.P. Garcí anda, C. Ló andpez and L. Entrena. *Extensive SEU Impact Analysis of a PIC Microprocessor for Selective Hardening.* Nuclear Science, IEEE Transactions on, vol. 57, no. 4, pages 1986 –1991, aug. 2010. doi:10.1109/TNS.2009.2039581. (Cited on page 49.)

[Vargas 1994] F. Vargas and M. Nicolaidis. *SEU-tolerant SRAM design based on current monitoring*. In Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on, pages 106–115, June 1994. doi:10.1109/FTCS.1994.315652. (Cited on pages 150 and 152.)

[Wang 2004] N.J. Wang, J. Quek, T.M. Rafacz and S.J. Patel. *Characterizing the effects of transient faults on a high-performance processor pipeline*. In Dependable Systems and Networks, 2004 International Conference on, pages 61–70, June 2004. doi:10.1109/DSN.2004.1311877. (Cited on page 93.)

[Wang 2011] Feng Wang and Yuan Xie. *Soft Error Rate Analysis for Combinational Logic Using an Accurate Electrical Masking Model*. Dependable and Secure Computing, IEEE Transactions on, vol. 8, no. 1, pages 137–146, 2011. doi:10.1109/TDSC.2009.29. (Cited on page 93.)

[Wasserman 2005] Larry A. Wasserman. All of statistics. Springer Texts in Statistics. Springer Verlag, 2005. (Cited on pages 71, 78 and 143.)

[Weaver 2004] C.T. Weaver, J. Emer, S.S. Mukherjee and S.K. Reinhardt. *Reducing the soft-error rate of a high-performance microprocessor*. Micro, IEEE, vol. 24, no. 6, pages 30–37, 2004. doi:10.1109/MM.2004.86. (Cited on page 47.)

[Wen 2010a] Shi-Jie Wen, S. Y. Pai, R. Wong, M. Romain and N. Tam. *B10 finding and correlation to thermal neutron soft error rate sensitivity for SRAMs in the sub-micron technology*. In Integrated Reliability Workshop Final Report (IRW), 2010 IEEE International, pages 31–33, 2010. doi:10.1109/IIRW.2010.5706480. (Cited on page 6.)

[Wen 2010b] ShiJie Wen, R. Wong, M. Romain and N. Tam. *Thermal neutron soft error rate for SRAMS in the 90NM to 45NM technology range*. In Reliability Physics Symposium (IRPS), 2010 IEEE International, pages 1036–1039, 2010. doi:10.1109/IRPS.2010.5488681. (Cited on page 6.)

[Wikipedia 2014] Wikipedia [online]. 2014. URL: http://en.wikipedia.org/wiki/Moore's_law [accessedJanuary 2014]. (Cited on page 16.)

[Wirth 2008] G. Wirth. *Bulk built in current sensors for single event transient detection in deep-submicron technologies*, 2008. (Cited on page 150.)

[Wong 2012] R. Wong, B.L. Bhuva, A. Evans and S. Wen. *System-level reliability using component-level failure signatures*. In Reliability Physics Symposium (IRPS), 2012 IEEE International, pages 4B.3.1–4B.3.7, April 2012. doi:10.1109/IRPS.2012.6241832. (Cited on page 105.)

[Wrobel 2000] F. Wrobel, J.-M. Palau, M. C. Calvet, O. Bersillon and H. Duarte. *Incidence of multi-particle events on soft error rates caused by n-Si nuclear*

reactions. Nuclear Science, IEEE Transactions on, vol. 47, no. 6, pages 2580–2585, Dec 2000. doi:10.1109/23.903812. (Cited on page 4.)

[Wrobel 2003] F. Wrobel, J.-M. Palau, M.-C. Calvet and P. Iacconi. *Contribution of SiO2 in neutron-induced SEU in SRAMs*. Nuclear Science, IEEE Transactions on, vol. 50, no. 6, pages 2055–2059, Dec 2003. doi:10.1109/TNS.2003.821596. (Cited on page 4.)

[Wunderlich 1985] H. Wunderlich. *PROTEST: A Tool for Probabilistic Testability Analysis*. In Design Automation, 1985. 22nd Conference on, pages 204–211, June 1985. doi:10.1109/DAC.1985.1585936. (Cited on page 60.)

[Xie 2014] Hao Xie, Li Chen, Adrian Evans and Shi-Jie Wen. *Algorithm for Fast Synthesis of Redundant Combinatorial Logic for Selective Fault Tolerance*. In SELSE Workshop, 2014. (Cited on page 139.)

[Xilinx 2012] Xilinx. Xapp538 soft error mitigation using prioritized essential bits [online]. 2012. URL: http://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf [accessedApril 2014]. (Cited on page 149.)

[Yu 2005] Yangyang Yu, B. Bastien and B.W. Johnson. *A state of research review on fault injection techniques and a case study*. In Reliability and Maintainability Symposium, 2005. Proceedings. Annual, pages 386 – 392, 24-27, 2005. doi:10.1109/RAMS.2005.1408393. (Cited on page 49.)

[Yu 2009] Hai Yu, M. Nicolaidis and L. Anghel. *An effective approach to detect logic soft errors in digital circuits based on GRAAL*. In Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design, pages 236 –240, march 2009. doi:10.1109/ISQED.2009.4810300. (Cited on page 24.)

[Yu 2011] Hai Yu, M. Nicolaidis, L. Anghel and N.-E. Zergainoh. *Efficient Fault Detection Architecture Design of Latch-Based Low Power DSP/MCU Processor*. In European Test Symposium (ETS), 2011 16th IEEE, pages 93 –98, may 2011. doi:10.1109/ETS.2011.20. (Cited on page 24.)

[Yuan 2013] Feng Yuan and Qiang Xu. *InTimeFix: A low-cost and scalable technique for in-situ timing error masking in logic circuits*. In Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE, pages 1–6, May 2013. (Cited on page 139.)

[Zhang 2006a] Bin Zhang, Wei-Shen Wang and M. Orshansky. *FASER: fast analysis of soft error susceptibility for cell-based designs*. In Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on, pages 6 pp.–760, 2006. doi:10.1109/ISQED.2006.64. (Cited on pages 57 and 93.)

[Zhang 2006b] Ming Zhang, S Mitra, T. M. Mak, N. Seifert, N.J. Wang, Quan Shi, Kee Sup Kim, N.R. Shanbhag and S.J. Patel. *Sequential Element Design*

With Built-In Soft Error Resilience. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, no. 12, pages 1368–1378, Dec 2006. doi:10.1109/TVLSI.2006.887832. (Cited on page 26.)

[Zhang 2010] Zhichao Zhang, Tao Wang, Li Chen and Jinsheng Yang. *A new Bulk Built-In Current Sensing circuit for single-event transient detection*. In Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on, pages 1–4, May 2010. doi:10.1109/CCECE.2010.5575124. (Cited on page 150.)

[Zhang 2011] M. Zhang, J. Park, G. Kim, K. Kim, R. Gaillard, E. Schaefer and O. Lauzeral. *Thermal Neutron SER Testing and Analysis: Findings from a 32nm HKMG SRAM Case Study*. In SELSE Workshop, Volume 7. SELSE Workshop, 2011. (Cited on page 6.)

[Zhang 2013] Zhichao Zhang, Yi Ren, Li Chen, NelsonJ. Gaspard, Arthur.F. Witulski, TimothyW. Holma n, BharatL. Bhuva, Shi-Jie Wen and Ramaswami Sammynaiken. *A Bulk Built-In Voltage Sensor to Detect Physical Location of Single-Event Transients*. Journal of Electronic Testing, vol. 29, no. 2, pages 249–253, 2013. URL: http://dx.doi.org/10.1007/s10836-013-5364-1, doi:10.1007/s10836-013-5364-1. (Cited on page 150.)

[Zheng 2009] Yexin Zheng and Chao Huang. *Defect-aware logic mapping for nanowire-based programmable logic arrays via satisfiability*. In Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09., pages 1279–1283, April 2009. doi:10.1109/DATE.2009.5090862. (Cited on page 140.)

[Ziegler 1979] J.F. Ziegler and W.a. Lanford. *Effect of Cosmic Rays on Computer Memories*, 1979. (Cited on page 6.)

[Ziegler 1981] J.F. Ziegler and W.A. Lanford. *The effect of sea level cosmic rays on electronic devices*. Journal of Applied Physics, vol. 52, no. 6, pages 4305–4312, 1981. doi:10.1063/1.329243. (Cited on page 4.)

[Ziegler 1996] J.F. Ziegler, H.W. Curtis, H.P. Muhfeld and C.J. Montrose. *IBM Experiments in Soft Fails in Computer Electronics (1978-1994)*, 1996. (Cited on pages 4, 6 and 7.)

Abstraction Techniques for Scalable Soft Error Analysis and Mitigation

Abstract: The main objective of this thesis is to develop techniques that can be used to analyze and mitigate the effects of radiation-induced soft errors in industrial scale integrated circuits. To achieve this goal, several methods have been developed based on analyzing the design at higher levels of abstraction. These techniques address both sequential and combinatorial SER.

Fault-injection simulations remain the primary method for analyzing the effects of soft errors. In this thesis, techniques which significantly speed-up fault-injection simulations are presented. Soft errors in flip-flops are typically mitigated by selectively replacing the most critical flip-flops with hardened implementations. Selecting an optimal set to harden is a compute intensive problem and the second contribution consists of a clustering technique which significantly reduces the number of fault-injections required to perform selective mitigation.

In terrestrial applications, the effect of soft errors in combinatorial logic has been fairly small. It is known that this effect is growing, yet there exist few techniques which can quickly estimate the extent of combinatorial SER for an entire integrated circuit. The third contribution of this thesis is a hierarchical approach to combinatorial soft error analysis.

Systems-on-chip are often developed by re-using design-blocks that come from multiple sources. In this context, there is a need to develop and exchange reliability models. The final contribution of this thesis consists of an application specific modeling language called RIIF (Reliability Information Interchange Format). This language is able to model how faults at the gate-level propagate up to the block and chip-level. Work is underway to standardize the RIIF modeling language as well as to extend it beyond modeling of radiation-induced failures.

In addition to the main axis of research, some tangential topics were studied in collaboration with other teams. One of these consisted in the development of a novel approach for protecting ternary content addressable memories (TCAMs), a special type of memory important in networking applications. The second supplemental project resulted in an algorithm for quickly generating approximate redundant logic which can protect combinatorial networks against permanent faults. Finally an approach for reducing the detection time for errors in the configuration RAM for Field-Programmable Gate-Arrays (FPGAs) was outlined.

Keywords: Single-event effects, single-event upsets, single-event transients, reliable systems, fault-injection